Electrosoft

MS Cryptographic Service Provider

Design,

Build and Installation Procedure

Prepared for:

NIST

Date: 10/21/2009
Version: 1.3
(**Draft**)

Electrosoft
11417 Sunset Hills Road, Suite 228
Reston, VA 20190
(703) 953-1017
http://www.electrosoft-inc.com

Tables of Contents

List of Figures

List of Tables

1. Introduction

1.1 Background

The National Institute of Standards and Technology (NIST) Federal Information Processing Standard 201 (FIPS 201) - *Personal Identity Verification (PIV) of Federal Employees and Contractors*, was developed to establish standards for a secure and reliable form of identity credentials. FIPS 201 along with its associated publications provide detailed specifications for Federal agencies and departments, in order for them to deploy PIV Cards to their personnel.

FIPS 201 is accompanied by three main documents:

1 Special Publication 800-73-1 (NIST SP 800-73-1) - *Interfaces for Personal Identity Verification*, specifies interface requirements for retrieving and using the identity credentials from the PIV Card. It also defines a PIV data model, which details the structure and format of the information stored on the PIV Card;

2 Special Publication 800-76 (NIST SP 800-76) - *Biometric Data Specification for Personal Identity Verification,* contains technical specifications for biometric data mandated in FIPS 201; and

3 Special Publication 800-78 (NIST SP 800-78) - *Cryptographic Algorithms and Key Sizes for Personal Identity Verification*, specifies the cryptographic algorithms and key sizes for performing cryptographic operations on PIV data objects defined as part of the PIV data model.

Once implemented and deployed by the various Federal agencies, the PIV card is envisioned to provide the attributes of security, authentication, trust and privacy using this commonly accepted identification credential.

1.2 Purpose and Scope

Microsoft provides an Application Programming Interface (API) called Crypto API (CAPI) to support cryptographic and PKI operations. Microsoft CAPI works with a number of Cryptographic Service Providers (CSPs) that contain actual implementations of cryptographic standards and algorithms. Different vendors develop various CSPs that have different cryptographic capabilities.

The main purpose of this document is to describe the design of the NIST-ESI CSP and how the CryptoSPI functions have been implemented. The primary and only intent of this CSP is to perform Windows Logon using a PIV Card or the PIV Reference Implementation. In this respect, the NIST-ESI CSP uses the SP 800-73-1 Client API for all communications with the PIV Card or PIV Reference Implementation.

This document also provides details on the sequence of function calls that are made to the NIST-ESI CSP for logging into a Window's domain using smart cards. It goes on to describe the components that are responsible for interactive logon and the communications that occur between these components.

1.3 Assumptions

As mentioned earlier, the NIST-ESI CSP has been developed solely for demonstrating Windows Smart Card Logon. Considering the intent of the NIST-ESI CSP, it is currently not capable of performing any

enrollment functions (i.e. generating key pairs and requesting corresponding certificates from a Certification Authority (CA)). This CSP assumes that the PIV Card/PIV Reference Implementation has been populated with the necessary credentials (PIV Authentication Key and Certificate for Windows Logon) and the PIV Authenticate Certificate profile is compliant with the certificate profile required for MS Windows Smart Card Logon.

1.4 Document Organization

The structure of the document is as follows:

- Section 1, Introduction, provides an introduction to this document and includes the purpose, scope and assumptions with which this document has been developed.

- Section 2, Architecture and Design of the NIST-ESI Cryptographic Service Provider, provides a brief introduction of the Microsoft Cryptographic API Architecture, and provides the reader with the background of the role of a CSP within this architecture. The latter half of this section goes into the details of the design and implementation of the NIST-ESI CSP.

- Section 3, Functional Process for Windows Logon using Smart Cards, describes briefly the components that are responsible for an interactive logon using a smart card, and discusses the interaction amongst these components.

- Section 4, NIST-ESI CSP Build and Installation Procedure, discusses the build and installation procedure for the Cryptographic Service Provider.

- Appendix A, Acronyms and Abbreviations, lists the full form of acronyms and abbreviations used in this document.

- Appendix B, References, provides a list of reference material used to develop this document.

2. Architecture and Design of the NIST-ESI Cryptographic Service Provider

2.1 Microsoft CSP Architectural Overview

The Microsoft Cryptographic API contains functions that allow applications to encrypt or digitally sign data in a flexible manner, while providing protection for the user's sensitive key data. As described above, these cryptographic operations are performed by independent modules known as cryptographic service providers.

A *Cryptographic Service Provider* contains implementations of cryptographic standards and algorithms. At a minimum, a CSP consists of a *dynamic-link library* (DLL) that implements the functions in CryptoSPI (a *system program interface*). Many CSPs contain the implementation of all of the CryptoSPI functions in software. Some CSPs, however, implement the CryptoSPI functions in hardware, such as a *smart card* or secure coprocessor. Sometimes, a CSP does not implement its own functions, but acts as a pass-through layer, facilitating the communication between the operating system and the module that actually implements the functions.

Applications do not communicate directly with a CSP. Instead, applications call CryptoAPI functions exposed by the operating system's Advapi32.dll and Crypt32.dll files. The operating system filters these function calls and passes them on to the appropriate CSP functions through CryptoSPI. These applications use *handles* to refer to data objects within the CSP. Objects that are referenced by their handles include key containers, hash objects, session key objects, and public/private key pair objects.

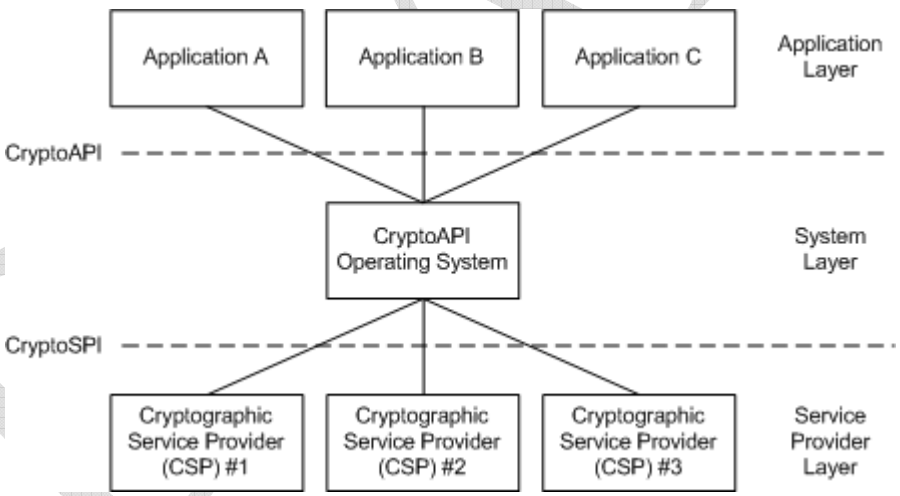

Figure 1 - Invocation of CSPs by Applications

Custom CSPs can be developed to work within the Microsoft environment. Custom CSPs must support all of the functions within the CryptoSPI; each function corresponds directly to a CryptoAPI cryptographic function. Custom CSPs must support the following DLL entry points:

No.	CryptoSPI Function Name
1.	CPAcquireContext
2.	CPReleaseContext
3.	CPDecrypt
4.	CPImportKey

5.	CPDestroyKey
6.	CPGenRandom
7.	CPGetKeyParam
8.	CPSetKeyParam
9.	CPGetUserKey
10.	CPGetProvParam
11.	CPHashData
12.	CPCreateHash
13.	CPDestroyHash
14.	CPSignHash
15.	CPSetHashParam
16.	CPSetProvParam
17.	CPVerifySignature
18.	CPGetHashParam
19.	CPHashSessionKey
20.	CPDuplicateHash
21.	CPDupicateKey
22.	CPDeriveKey
23.	CPExportKey
24.	CPGenKey
25.	CPEncrypt

Table 1 - CryptoSPI Functions

2.2 Application Call Stack using the NIST-ESI CSP

Figure 2 depicts the layers involved when the Windows Logon application invokes the NIST-ESI CSP to utilize credentials loaded on a PIV Card. The NIST-ESI CSP is used by the Windows Logon application to obtain cryptographic services in support of the mutual authentication that needs to occur for a successful logon using the PIV Card.

The NIST-ESI CSP interfaces with the PIV Middleware, which may be configured to invoke the PC/SC interface to exchange APDUs with a PIV Card. In this configuration, the smart card reader driver and I/O drivers are part of the call stack.

Alternately, the PIV Middleware may be configured to invoke the PC/SC interface to exchange APDUs with the PIV Card Simulator. This simulator provides a reference implementation of the PIV interfaces, runs in a Java Runtime environment, and uses a loadable file system. In this configuration, the NIST Virtual Smart Card Reader (VSCR) driver and service are part of the call stack.

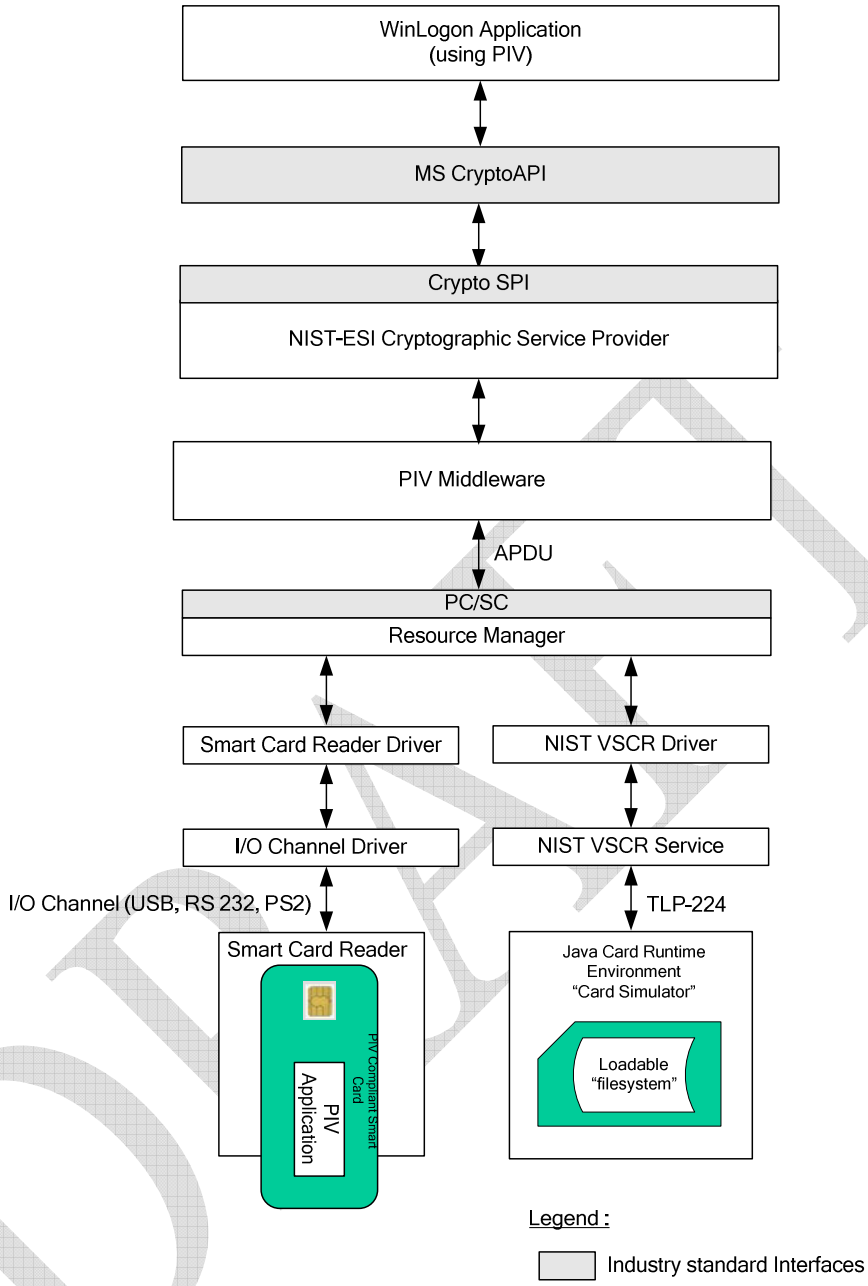

Figure 2 – Windows Smart Card Logon Call Stack for PIV

2.3 Invoked CryptoSPI functions during Windows Smart Card Logon

Table 2 lists the CryptoSPI functions that are invoked during Windows Smart Card Logon:

No.	CryptoSPI Functions	Required for Windows Smart Card Logon
1.	CPAcquireContext	X
2.	CPReleaseContext	X
3.	CPDecrypt	X

4.	CPImportKey	X
5.	CPDestroyKey	X
6.	CPGenRandom	X
7.	CPGetKeyParam	X
8.	CPSetKeyParam	X
9.	CPGetUserKey	X
10.	CPGetProvParam	X
11.	CPHashData	X
12.	CPCreateHash	X
13.	CPDestroyHash	X
14.	CPSignHash	X
15.	CPSetHashParam	X
16.	CPSetProvParam	X
17.	CPVerifySignature	
18.	CPGetHashParam	
19.	CPHashSessionKey	
20.	CPDuplicateHash	
21.	CPDupicateKey	
22.	CPDeriveKey	
23.	CPExportKey	
24.	CPGenKey	
25.	CPEncrypt	

Table 2 - CryptoSPI Functions Required to support Windows Smart Card Logon

2.4 High Level Design of the NIST-ESI CSP

The NIST-ESI CSP was developed primarily to support the use of a PIV Card for Windows Smart Card Logon. Hence, this CSP provides full functionality implementation of only those CryptoSPI functions that are required to enable Windows Smart Card Logon as per Table 2 . The remaining CryptoSPI functions are implemented as stub functions only.

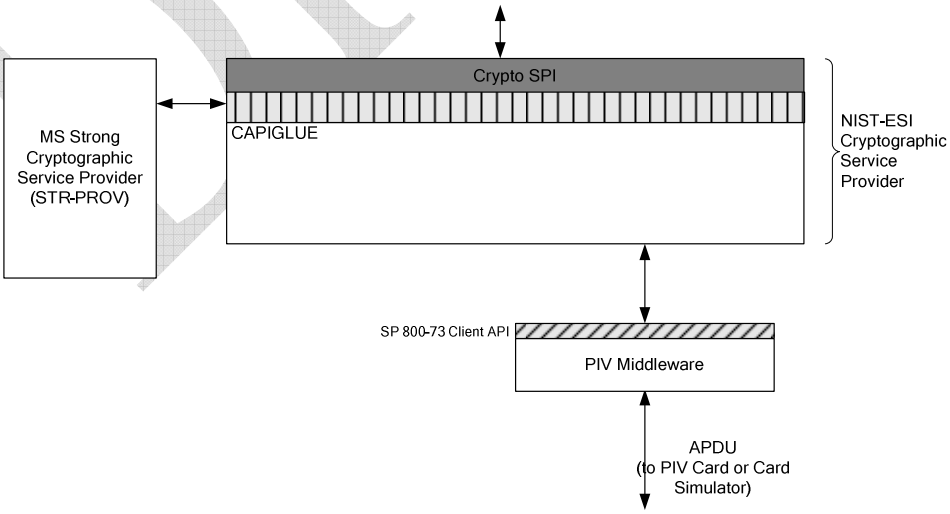

Figure 3 - NIST-ESI CSP Design

Figure 3 provides a graphic description of the high level design of the NIST-ESI CSP. The CAPIGLUE layer code handles calls to each of the CryptoSPI functions. The PIV Middleware represents a dynamic linked library that implement the PIV Client API (as specified in NIST SP 800-73-1) and sends out APDUs to a PIV Card or PIV Card Simulator using the PIV Card Command Interface (also specified in NIST SP 800-73-1). The Microsoft Strong Cryptographic Service Provider (STR-PROV) is used by the NIST-ESI CSP for performing various cryptographic operations in software.

The interactions between the components described in Figure 3 are as follows. The CAPIGLUE layer performs all of the non-cryptographic operations. However, when cryptographic operations are required, it either invokes the PIV Middleware or the STR-PROV. For all cryptographic operations that require use of the PIV authentication key (e.g. signature using a private key, private key decryption), the CAPIGLUE layer calls the PIV Middleware. For other cryptographic operations (e.g., cryptographic hash, public key encryption, symmetric key operations etc.) the CAPIGLUE layer invokes the STR-PROV. The CAPIGLUE and PIV Middleware do not implement any cryptographic algorithms; the STR-PROV and the PIV Card are the modules where all cryptographic algorithms are implemented.

2.5 CryptoSPI function sequence for Windows Logon

This section describes the sequence of function calls made to the CSP during Windows Smart Card Logon. This sequence has been captured by tracing the function calls made to the NIST-ESI CSP.

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
1.	CPAcquireContext	▪ Acquire a handle to the default container on the card ▪ Returns handle to default container context	▪ Connect to and Select the PIV Card Application for the first time and store handle ▪ Acquire Context to STR-PROV
2.	CPGetUserKey	▪ Get a handle to the key exchange key pair in the default container.	▪ Gets a handle to the PIV Authentication Certificate
3.	CPGetKeyParam	▪ Called to retrieve the size of the certificate on the card (using the context acquired in #1) ▪ Returns the certificate size in bytes	▪ Gets and returns the size of the PIV Authentication Certificate
4.	CPGetKeyParam	▪ Get the certificate corresponding to the exchange key from the card	▪ Gets the PIV Authentication Certificate
5.	CPGetProvParam	▪ Called with the PP_CONTAINER flag to retrieve the size of the name of the current	▪ Returns the size of an arbitrary string created for the name of the container

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
		container (using the context acquired in #1) ▪ Returns the size of the name of the current container in bytes	
6.	CPGetProvParam	▪ Called with the PP_CONTAINER flag to retrieve the name of the current container (using the context acquired in #1) ▪ Returns the name of the current container	▪ Returns the arbitrary container name string
7.	CPGetProvParam	▪ Called with the PP_NAME flag to retrieve the size of the name of the CSP (using the context acquired in #1) ▪ Returns the size of the name of the CSP in bytes	▪ Returns the size of the name of the NIST-ESI CSP.
8.	CPGetProvParam	▪ Called with the PP_NAME flag to retrieve the name of the CSP (using the context acquired in #1) ▪ Returns the name of the CSP	▪ Returns the name of the CSP
9.	CPDestroyKey	▪ Release the handle to the key exchange pair in the default container (using the context acquired in #1)	▪ Release the handle to the PIV Authentication Certificate obtained in #2
10.	CPAcquireContext	▪ Acquire a handle to the default container on the card ▪ Returns handle to default container context	▪ Returns the handle previously created in #1 ▪ Acquire Context to STR-PROV
11.	CPGetProvParam	▪ Called with the PP_ENUMALGS parameter and the CRYPT_FIRST flag to	▪ The NIST-ESI CSP declares that it can handle DES, 3DES, RC2 and RC4 algorithms.

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
		get information about the first algorithm supported by the CSP (card) (using the context acquired in #10) ▪ Returns a PROV_ENUMALGS structure filled out with information about the first algorithm	▪ This function is called repeatedly to get the ALGO structure
12.	CPSetProvParam	▪ Present the PIN to the card (using the context acquired in #10)	▪ Log into the PIV Card Application using the PIN presented.
13.	CPCreateHash	▪ Get a handle to an MD5 hash object (using the context acquired in #1) ▪ Returns a handle to a hash object	▪ Call CryptCreateHash in STR-PROV to create and initiate the hash ▪ Return the handle of the hash object
14.	CPHashData	▪ Hash the data passed in (using the context acquired in #1)	▪ Call CryptHashData in STR-PROV
15.	CPSignHash	▪ Sign the hash (using the context acquired in #1)	▪ Call pivCrypt to sign the data with key reference 9A
16.	CPDestroyHash	▪ Release the handle to the hash object	▪ Release the handle to the hash object created in #13
17.	CPSetProvParam	▪ Present the PIN to the card (using the context acquired in #10)	▪ Log into the PIV Card Application using the PIN presented. ▪ If the cached PIN matches the presented PIN then a logging into the card application is not performed.
18.	CPCreateHash	▪ Get a handle to an MD5 hash object (using the context acquired in #1) ▪ Returns a handle to a hash object	▪ Call CryptCreateHash in STR-PROV to create and initiate the hash ▪ Return the handle of the hash object
19.	CPHashData	▪ Hash the data passed in (using the context acquired in #1)	▪ Call CryptHashData in STR-PROV

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
20.	CPGetHashParam	▪ Called with HP_HASHVAL to retrieve the size of the hash value (using the context acquired in #1) ▪ Returns the size of the hash value in bytes	▪ Call CryptGetHashParam in STR-PROV to return the size of the hash value
21.	CPGetHashParam	▪ Called with HP_HASHVAL to retrieve the hash value (using the context acquired in #1) ▪ Returns the hash value	▪ Call CryptGetHashParam in STR-PROV to return the hash value
22.	CPCreateHash	▪ Get a handle to an MD5 hash object (using the context acquired in #1) ▪ Returns a handle to a hash object	▪ Call CryptCreateHash in STR-PROV to create and initiate the hash ▪ Return the handle of the hash object
23.	CPSetHashParam	▪ Called with the HP_HASHVAL parameter and no data (using the context acquired in #1)	▪ Call CryptSetHashParam in STR-PROV
24.	CPSignHash	▪ Call SignHash to get the size of the signature (using the context acquired in #1) (for hash created in #22) ▪ Returns the size of the signature in bytes	▪ Call pivCrypt to sign the data with key reference 9A ▪ Calculate the signature length and return the length of the signature
25.	CPSignHash	▪ Sign the hash signature (using the context acquired in #1) (for hash created in #22) ▪ Returns the signed hash	▪ Call pivCrypt to sign the data with key reference 9A
26.	CPDestroyHash	▪ Release the handle to the hash object (using the context acquired in #1) (for hash created in #22)	▪ Release the handle to the hash object created in #22
27.	CPDestroyHash	▪ Release the handle to the	▪ Release the handle to the

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
		hash object (using the context acquired in #1) (for hash created in #18)	hash object created in #18
28.	CPSetProvParam	▪ Present the PIN to the card (using the context acquired in #10)	▪ Log into the PIV Card Application using the PIN presented. ▪ If the cached PIN matches the presented PIN then a logging into the card application is not performed.
29.	CPSetProvParam	▪ Present the PIN to the card (using the context acquired in #10)	▪ Log into the PIV Card Application using the PIN presented. ▪ If the cached PIN matches the presented PIN then a logging into the card application is not performed.
30.	CPGetUserKey	▪ Get a handle to the key exchange key pair in the default container.	▪ Gets a handle to the PIV Authentication Certificate
31.	CPImportKey	▪ Import a key (using the context acquired in #1)	▪ Encrypted session key is passed into NIST-ESI CSP. ▪ NIST-ESI CSP decrypts the encrypted session key by using pivCrypt (using key reference 9A) ▪ The unencrypted session key is imported into the STR-PROV by: 1. Creating an asymmetric key pair in STR-PROV 2. Encrypting the session key with the generated public key 3. Finally this encrypted key is structured into a BLOB and imported into the

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
			STR-PROV
32.	CPSetKeyParam	▪ Set the initialization vector to all zeroes (using the context acquired in #1)	▪ Call CryptSetKeyParam in STR-PROV and pass in a handle to the key obtained in #31
33.	CPDestroyKey	▪ Release the handle to the key	▪ Release the handle to key obtained in #31
34.	CPDecrypt	▪ Decrypt a chunk of data with the imported key (using the context acquired in #1)	▪ Call CryptDecrypt in STR-PROV to perform decryption with the help of the symmetric key imported in #31
35.	CPDestroyKey	▪ Release the handle to the key exchange pair in the default container (using the context acquired in #1)	▪ Release the handle to the PIV Authentication Certificate obtained in #30
36.	CPGenRandom	▪ Generate 32 random bytes (using the context acquired in #10)	▪ Call CryptGenRandom in STR-PROV to generate the random bytes
37.	CPGenRandom	▪ Generate 32 random bytes (using the context acquired in #10)	▪ Call CryptGenRandom in STR-PROV to generate the random bytes
38.	CPCreateHash	▪ Get a handle to a hash object (using the context acquired in #10) ▪ Returns a handle to a hash object	▪ Call CryptCreateHash in STR-PROV to create and initiate the hash ▪ Return the handle of the hash object
39.	CPHashData	▪ Hash the data passed in (using the context acquired in #10)	▪ Call CreateHashData in STR-PROV
40.	CPSetProvParam	▪ Present the PIN to the card (using the context acquired in #10)	▪ Log into the PIV Card Application using the PIN presented. ▪ If the cached PIN matches the presented PIN then a logging into the card application is not performed.

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
41.	CPSignHash	▪ Call SignHash to get the size of the signature (using the context acquired in #10) ▪ Returns the size of the signature in bytes	▪ Call pivCrypt to sign the data with key reference 9A ▪ Calculate the signature length and return the length of the signature
42.	CPSignHash	▪ Sign the hash (using the context acquired in #10) (for hash created in #38) ▪ Returns the signed hash	▪ Call pivCrypt to sign the data with key reference 9A
43.	CPDestroyHash	▪ Release the handle to the hash object (using the context acquired in #10)	▪ Release the handle to the hash object created in #38
44.	CPCreateHash	▪ Get a handle to a hash object (using the context acquired in #10) ▪ Returns a handle to a hash object	▪ Call CryptCreateHash in STR-PROV to create and initiate the hash ▪ Return the handle of the hash object
45.	CPHashData	▪ Hash the data passed in (using the context acquired in #10)	▪ Call CryptHashData in STR-PROV
46.	CPSetProvParam	▪ Present the PIN to the card (using the context acquired in #10)	▪ Log into the PIV Card Application using the PIN presented. ▪ If the cached PIN matches the presented PIN then a logging into the card application is not performed.
47.	CPSignHash	▪ Call SignHash to get the size of the signature (using the context acquired in #10) ▪ Returns the size of the signature in bytes	▪ Call pivCrypt to sign the data with key reference 9A ▪ Calculate the signature length and return the length of the signature
48.	CPSignHash	▪ Sign the hash (using the context acquired in #10) (for hash created in #44) ▪ Returns the signed hash	▪ Call pivCrypt to sign the data with key reference 9A

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
49.	CPDestroyHash	▪ Release the handle to the hash object (using the context acquired in #10)	▪ Release the handle to the hash object created in #44
50.	CPAcquireContext	▪ Acquire a handle to the default container on the card ▪ Returns handle to default container context	▪ Returns the handle previously created in #1
51.	CPGetProvParam	▪ Called with the PP_CONTAINER flag to retrieve the size of the name of the current container (using the context acquired in #52) ▪ Returns the size of the name of the current container in bytes	▪ Returns the size of an arbitrary string created for the name of the container
52.	CPGetProvParam	▪ Called with the PP_CONTAINER flag to retrieve the name of the current container (using the context acquired in #52) ▪ Returns the name of the current container	▪ Returns the arbitrary container name string
53.	CPAcquireContext	▪ Acquire a handle to the default container on the card ▪ Returns handle to default container context	▪ Returns the handle previously created in #1
54.	CPGetUserKey	▪ Get a handle to the key exchange key pair in the default container (using the context acquired in #53)	▪ Gets a handle to the PIV Authentication Certificate
55.	CPGetKeyParam	▪ Called to retrieve the size of the certificate on the card (using the context acquired in #53) ▪ Returns the certificate size in bytes	▪ Gets and returns the size of the PIV Authentication Certificate

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
56.	CPGetKeyParam	▪ Get the certificate corresponding to the exchange key from the card	▪ Gets the PIV Authentication Certificate
57.	CPDestroyKey	▪ Release the handle to the key exchange pair in the default container (using the context acquired in #53)	▪ Release the handle to the PIV Authentication Certificate obtained in #54
58.	CPGetUserKey	▪ Get a handle to the key exchange key pair in the default container (using the context acquired in #53)	▪ Gets a handle to the PIV Authentication Certificate
59.	CPGetKeyParam	▪ Called to retrieve the size of the certificate on the card (using the context acquired in #53) ▪ Returns the certificate size in bytes	▪ Gets and returns the size of the PIV Authentication Certificate
60.	CPGetKeyParam	▪ Get the certificate corresponding to the exchange key from the card	▪ Gets the PIV Authentication Certificate
61.	CPDestroyKey	▪ Release the handle to the key exchange pair in the default container (using the context acquired in #53)	▪ Release the handle to the PIV Authentication Certificate obtained in #58
62.	CPGetProvParam	▪ Called with the PP_CONTAINER flag to retrieve the size of the name of the current container ▪ Returns the size of the name of the current container in bytes ▪	▪ Returns the size of an arbitrary string created for the name of the container
63.	CPReleaseContext	▪ Release the handle to the default container	▪ Verify if the handle is null, if yes, then call pivDisconnect.

Sr. No.	CSP Function	General Purpose	NIST-ESI CSP Implementation
64.	CPReleaseContext	▪ Release the handle to the default container	▪ Disconnect from the PIV Application and Release the handle to STR-PROV.

2.6 Detailed Design of the NIST-ESI CSP

This section describes the detailed design of the NIST-ESI CSP. The subsection below provides the high-level implementation details for the CAPIGLUE implementation of each CryptoSPI function within the NIST-ESI CSP.

Table 3 lists the entry points on the PIV client-application programming interface, as implemented by the PIV Middleware. The NIST SP 800-73-1 document provides full detail on these interfaces. These interfaces are invoked by CAPIGLUE to obtain access to the credentials on the PIV Card.

Type	Name
Entry Points for Communication	pivConnect
	pivDisconnect
Entry Points for Data Access	pivSelectCardApplication
	pivLogIntoCardApplication
	pivGetData
	pivLogoutOfCardApplication
Entry Points for Cryptographic Operations	pivCrypt
Entry Points for Credential Initialization and Administration[1]	pivPutData
	pivGenerateKeyPair

Table 3 - Entry Points on PIV Client Application Programming Interface

[1] The Entry points for credential initialization and Administration have not been used by the NIST-ESI CSP, since this functionality is currently beyond the scope of the functionality of the CSP.

2.6.1 CryptoSPI Functions Implementation Descriptions

The following section describes the details for the CryptoSPI functions implemented by the NIST-ESI CSP.

2.6.1.1 CPAcquireContext

Description:	Used to acquire a context handle to a CSP and the key container
NIST-ESI CSP Implementation:	• Call *pivConnect* and *pivSelectCardApplication* from PIV Middleware • Call *CryptAcquireContext* on STR-PROV and store the handle

2.6.1.2 CPReleaseContext

Description:	Used to release the context handle to the CSP created by *CPAcquireContext* *Note:* Key and Hash handles associated with the CSP handle cannot be used after calling this function
NIST-ESI CSP Implementation:	• Call *pivDisconnect* on the PIV Middleware • Call *CryptReleaseContext* on STR-PROV

2.6.1.3 CPDecrypt

Description:	Used to decrypt the cipher text
NIST-ESI CSP Implementation:	• Call *CryptDecrypt* on STR-PROV to perform decryption

2.6.1.4 CPImportKey

Description:	Imports a cryptographic key into the CSP key container
NIST-ESI CSP Implementation:	• Decrypt the data passed in with a call to *pivCrypt* on the PIV Middleware • Imports the encrypted key • Encrypted session key is passed into NIST-ESI CSP. • NIST-ESI CSP decrypts the encrypted session key by using pivCrypt

	(using key reference 9A)
	• The unencrypted session key is imported into the STR-PROV by:
	  o Creating an asymmetric key pair in STR-PROV
	  o Encrypting the session key with the generated public key
	  o Finally this encrypted key is structured into a BLOB and imported into the STR-PROV

2.6.1.5 CPDestroyKey

Description:	Used to release the handle to the cryptographic key stored in the CSP key container. The key may no longer be used after this call
NIST-ESI CSP Implementation:	• Free the handle and release memory to keys in the NIST-ESI CSP • Call *CryptDestroyKey* on STR-PROV to destroy the key (in certain cases)

2.6.1.6 CPGenRandom

Description:	Used to fill a buffer with random bytes. The algorithm used is based on the secure hash standard random number generation
NIST-ESI CSP Implementation:	• Call *CryptGenRandom* on STR-PROV to generate the random bytes

2.6.1.7 CPGetKeyParam

Description:	Retrieves data that governs the operations of a key.
NIST-ESI CSP Implementation:	• The NIST-ESI CSP currently only handles type KP_CERTIFICATE • Copy the already fetched certificate blob (during CPGetUserKey) to the user buffer

2.6.1.8 CPSetKeyParam

Description:	Used to allow applications to customize various aspects of the operations of a key
NIST-ESI CSP Implementation:	• Call *CryptSetKeyParam* on STR-PROV and pass in a handle to the key

2.6.1.9 CPGetUserKey

Description:	Retrieves the handle to one of the user's key pairs in the key container
NIST-ESI CSP Implementation:	• Call *pivGetData* to get PIV Authentication Certificate on PIV Middleware

2.6.1.10 CPGetProvParam

Description:	Used to allow applications to get various aspects of a Cryptographic Service provider
NIST-ESI CSP Implementation:	• The NIST-ESI CSP passes the following data for the following flags: PP_CONTAINER - Pass hard coded value PP_ENUMALGS - Currently passing DES, 3DES, RC2 and RC4 PP_KEYEXCHANGE_PIN - Pass cached PIN if available PP_NAME - Pass hard coded value PP_PROVTYPE - Pass PROV_RSA_FULL PP_SIGNATURE_PIN - Pass cached PIN if available PP_VERSION - Pass Hard coded version

2.6.1.11 CPHashData

Description:	Used to add data into an existing hash.
NIST-ESI CSP Implementation:	• Call *CryptHashData* on STR-PROV to hash a string of bytes

2.6.1.12 CPCreateHash

Description:	Used to initiate the hashing of a stream of data.
NIST-ESI CSP Implementation:	• Call *CryptCreateHash* on STR-PROV to create and initiate the hash

2.6.1.13 CPDestroyHash

Description:	Used to destroy the hash object

NIST-ESI CSP Implementation:	• Call *CryptDestroyHash* on STR-PROV

2.6.1.14 CPSignHash

Description:	Used to create a digital signature from a hash
NIST-ESI CSP Implementation:	• Call *pivCrypt* on the PIV Middleware to sign the data with key reference '9A' • Call *CryptGetHashParam* on STR-PROV to get the size and value of the hash

2.6.1.15 CPSetHashParam

Description:	Used to allow applications to customize various aspects of the operations of a hash
NIST-ESI CSP Implementation:	• Call *CryptSetHashParam* on STR-PROV

2.6.1.16 CPSetProvParam

Description:	Used to allow applications to customize various aspects of the operations of a Cryptographic Service provider
NIST-ESI CSP Implementation:	• Call *pivLogIntoCardApplication* on the PIV Middleware to log into the card

2.6.1.17 CPGetHashParam

Description:	Used to retrieve data about the operations of a hash object. Also used to obtain the actual hash value
NIST-ESI CSP Implementation:	• Call *CryptGetHashParam* on STR-PROV

2.6.1.18 CPVerifySignature

Note:	This function has been stubbed in the NIST-ESI CSP, as it is not a necessary call for Windows logon

2.6.1.19 CPHashSessionKey

Note:	This function has been stubbed in the NIST-ESI CSP, as it is not a necessary call for Windows logon

2.6.1.20 CPDuplicateHash

Note:	This function has been stubbed in the NIST-ESI CSP, as it is not a necessary call for Windows logon

2.6.1.21 CPDuplicateKey

Note:	This function has been stubbed in the NIST-ESI CSP, as it is not a necessary call for Windows logon

2.6.1.22 CPDeriveKey

Note:	This function has been stubbed in the NIST-ESI CSP, as it is not a necessary call for Windows logon

2.6.1.23 CPExportKey

Note:	This function has been stubbed in the NIST-ESI CSP, as it is not a necessary call for Windows logon

2.6.1.24 CPGenKey

Note:	This function has been stubbed in the NIST-ESI CSP, as it is not a necessary call for Windows logon

2.6.1.25 CPEncrypt

Description:	Used to encrypt the cipher text

NIST-ESI CSP Implementation:	• Call *CryptEncrypt* on STR-PROV to perform encryption

3. Functional Process for Windows Logon using smart cards

This section describes the interactive logon architecture and the process of an interactive logon.

3.1 Components

Figure 4 provides an illustration of the components of the interactive logon process that are responsible for helping to establish secure user authentication.

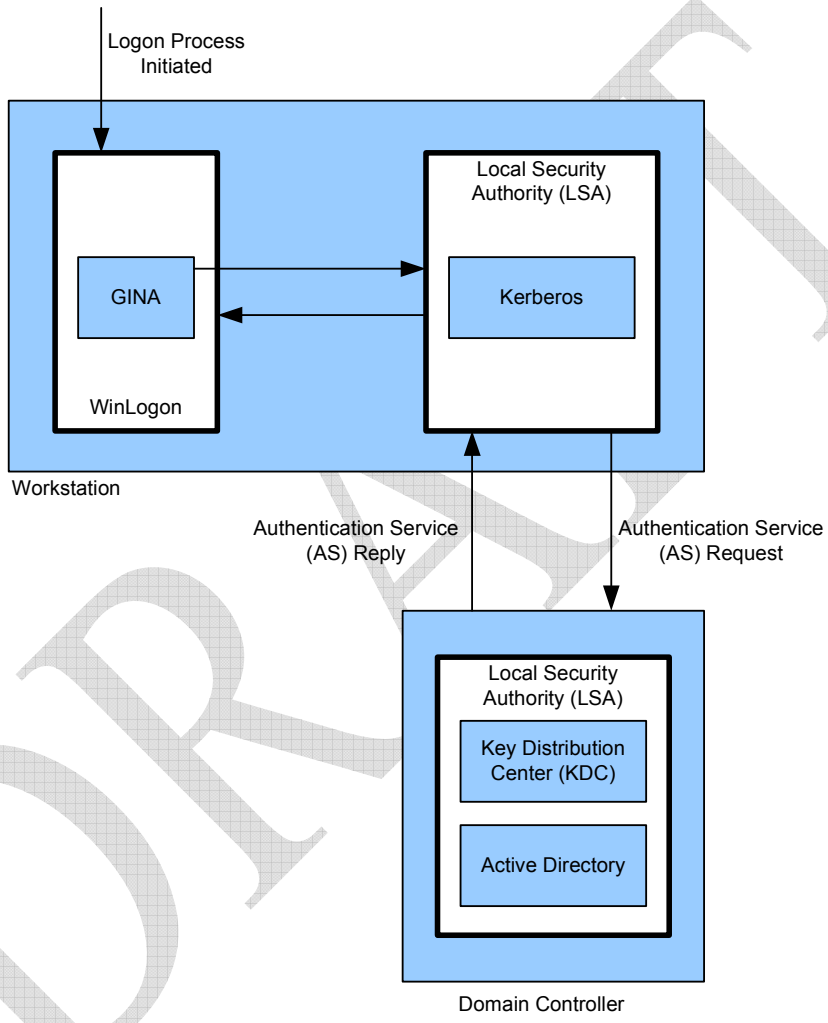

Figure 4 - Interactive Logon Components

The Windows Server 2003 interactive logon architecture components are explained in the following subsections.

3.1.1 Winlogon

Winlogon is the application responsible for managing secure user interactions. Winlogon initiates the logon process for Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0.

3.1.2 Graphical Identification and Authentication

Winlogon loads the Graphical Identification and Authentication (GINA) early in the boot process. GINA is a DLL module that operates in the security context of Winlogon, which is responsible for processing Secure Attention Sequence (SAS) events[2] and activating the user's shell.

The default MS GINA provided as part of Windows can be replaced to support specific and unique authentication methods. GINA customization is enabled to accommodate the use of authentication hardware tokens, such as retinal scanners and proprietary smart card solutions

3.1.3 Local Security Authority (LSA)

The LSA is a protected security subsystem that helps create secure user interactions within Windows Server 2003. Winlogon and the GINA call the LSA to process logon credentials.

The following components involved in user logon run in the security context of the LSA for logging on in a Windows Server 2003 domain environment:

- Authentication packages
- Active Directory

3.1.4 Authentication Package

Authentication packages on the user's local computer communicate with server authentication packages to authenticate users. The following table lists the Windows Server 2003 authentication packages used for interactive logons.

Name	Environment
Kerberos Version 5	Windows 2000, Windows XP, and Windows Server 2003
NTLM	Windows NT 4.0 and mixed environments

Table 4- Interactive Logon Authentication Packages

Note: There is an extension to the Kerberos version 5 protocol proposed by the IETF called PKINIT that allows for the use of a public key certificate in place of a password during the initial authentication. The PKINIT extension is the basis for smart card logon support in Windows 2003 and does not change the requirement for a long-term symmetric key used in the Kerberos Protocol.

3.2 Functional Process

In order to successfully logon to a Windows machine, the Local Security Authority (LSA) evaluates the user's credentials to determine if the logon should be processed as a logon to a local account or a logon to a domain account. If the authenticating domain controller is a computer running Windows 2000 Server or Windows Server 2003, the LSA will use Kerberos, the default authentication package for domain and

[2] The CTRL+ALT+DEL keystroke or the insertion of a smart card into the reader are examples of a secure attention sequence (SAS). Winlogon registers this sequence during the boot process to keep other programs and processes from using it.

network logons. The LSA uses NTLM to process domain logons in Windows NT 4.0 mixed environments. Since this document assumes an environment comprising of a Windows 2003 Server and a Windows XP client machine, details will be limited to the use of Kerberos.

The steps outlined in this section describe the activities that are carried out in order to authenticate the user to the Windows domain. Please note that these steps are not necessarily listed in the order in which they are performed as different functions are performed by different components. However, all of the steps need to occur in order for user identification and authentication into the domain environment.

The following steps occur at the client's machine:

- User inserts smart card into reader
- The Window Logon Service dispatches this event to MS GINA
- MS GINA prompts the user to enter a PIN
- After a user inputs a PIN to the logon dialog, Winlogon begins a sequence of actions to determine whether the user can be identified and authenticated based on credential information the user has provided (PIN and credentials on the smart card).
- The logon request first goes from the Windows Logon Service to the LSA that subsequently forwards it to the Kerberos authentication package running on the client.
- The Kerberos package sends an authentication service (AS) request to the Key Distribution Center (KDC) service running on a domain controller to request authentication and a Ticket Granting Ticket (TGT).
- As part of the AS request, the client-side Kerberos package includes the following items:
    - User Principal Name
    - The account domain name
    - Pre-authentication data
        - The user's X.509 version 3 certificate, retrieved from the smart card
        - An authenticator[3] (containing a timestamp) that is digitally signed by the user's private key so that the KDC can verify the AS request originated from the owner of the accompanying certificate.

The next set of steps occurs at the domain controller:

- Before the KDC can satisfy the AS request, it must first verify the certification path of the user's certificate to ensure that it can be trusted. The KDC uses CryptoAPI to build a certification path from the user's certificate to a root CA certificate residing in the system root store. If the KDC fails to build a valid certificate chain for any reason (e.g., the root certificate is not trusted, it cannot find parent certificates, the revocation status cannot be determined) the KDC will return an error and fail the request.

[3] The authenticator to be signed is first hashed using the MD-5 algorithm, which is the standard algorithm used during Windows Logon.

- The KDC must also verify that the issuing CA is authorized to issue certificates whose name information can be used as a basis for authentication within the domain. For this reason, the issuing CA must be published in the Active Directory.
- Upon successful verification of the user's certificate, the KDC then uses CryptoAPI to verify the digital signature on the authenticator that was included as signed data in the pre-authentication data fields. The signature verification is done using the public key from the user's certificate to prove that the request originated from the owner of the public key.
- After verifying the signature, the KDC service must then validate the timestamp in the authenticator to ensure the request is not a replay attack.
- Upon verifying the user's identity and that the certificate can be used to authenticate to the domain, the KDC service then queries the domain's directory for account information. The KDC service retrieves user account information from Active Directory based on the User Principal Name (UPN) specified in the Subject Alternative Name field in the user's public key certificate. The account information that the KDC retrieves from the directory is used to construct a TGT.
- The KDC creates credentials that the Kerberos client on the workstation can present to the ticket-granting service.
- The KDC signs the AS reply using its private key so that the client can verify the AS reply is from a trusted KDC.
- As part of the AS reply, the KDC encrypts the TGT using a generated logon session key. The logon session key is encrypted using the public key from the user's certificate. The encrypted key is then included in the pre-authentication data field of the KDC's Authentication Service (AS) response.
- The KDC also signs the TGT authorization data with the server's key (so that a rogue service cannot alter the authorization data after the TGT has been issued) and encrypts the TGT using the server's secret key[4].

The last set of steps occurs at the client:

- The client verifies the KDC's signature by first building a certification path from the KDC's certificate to a trusted root CA and then using the KDC's public key to verify the reply signature.
- The client, then, will extract the encrypted logon session key, decrypt it using the client's private key, and use the session key to decrypt the TGT. If the client is able to decrypt the TGT, then that proves to the KDC that the client was successfully able to get to the session key by decrypting it with its private key.
- Once in possession of the TGT, the standard Kerberos version 5 protocol is used to request tickets from the Ticket Granting Service (TGS) for other domain resources. Both the client and the KDC use the logon session key in all further communications with one another. There are no more interactions with the CSP.

[4] The client cannot read the Ticket Granting Ticket. Only KDC server's can read TGTs to secure access to user credentials, session keys, and other information.

4. NIST-ESI CSP Build and Installation Procedure

This section of the document describes the steps necessary to build and/or install the NIST-ESI CSP in order to be used for logging into the Windows domain. Please note that developers who wish to make changes to the code should follow the directions found in section 4.1. To demonstrate functionality of an already-built CSP, follow the directions found in section 4.2.

4.1 Build and Installation Procedure

The procedure outlined in this subsection needs to be followed in the event that the NIST-ESI CSP is used in a development environment. It assumes that the developer is familiar with utilizing Visual Studio .NET 2003.

4.1.1 Machine Prerequisite

In order for a Windows XP machine to use a CSP which has not been signed by Microsoft, a system file, advapi32.dll, must be patched and copied to the Windows system folder. This patched file has been provided as part of the NIST-ESI CSP. It is important to note that this file only works with Windows XP Professional Service Pack 2 (SP-2).

1. Unzip the nist-csp.zip file to a preferred location
2. Copy the advapi32.dl_ file in the .\Source\cspdk\WinXP\sp2 folder to the Windows\system32 folder
3. Re-boot the machine in Safe Mode
4. In the Windows\system32 folder, find the advapi32.dll system file and rename it to advapi32.bak
5. Rename the advapi32.dl_ file to advapi32.dll
6. Re-boot the machine in Normal mode

4.1.2 Build and Install for Windows Logon

This procedure assumes that the user has obtained a PIV smart card or the PIV Reference Implementation and populated it with the appropriate certificates prior to using the NIST-ESI CSP.

When using the PIV Reference Implementation (which consists of the PIV Card Simulator and Virtual Smart Card Reader), this procedure assumes that the user has previously built it. The "File_System_Initial_Configuration.inc" file of the PIV Card Simulator must be properly populated and the compiled PIV Card Simulator must be running on a separate machine which is accessible by the client machine on the local network. In addition, to invoke the NIST-ESI CSP at logon, the Virtual Smart Card Reader must be configured on the client machine to insert a virtual smart card at the logon prompt, which will trigger MS GINA to initiate the logon sequence. The PIV Reference Implementation must also be associated with the NIST-ESI CSP. Instructions for this procedure can be found in Section 4.3.

1. In the .\Source\dev\escsp folder, open the workspace escsp.sln
2. Build the entire solution in Release mode
3. Create a folder named "csp-setup" on the local machine (e.g., c:\csp-setup)
4. Copy the contents of the .\Binaries folder to the newly created folder

5. Copy the escsp.dll file located in the .\Source\dev\escsp\Release folder to the newly created folder
6. Copy the escspinstall.exe file located in the .\Source\dev\escsp\escspinstall\Release folder to the newly created folder
7. Run the prepcsp.bat file located in the newly created folder
8. Select option 1 "Install NIST CSP for PIV Card"
9. A message is displayed indicating installation is complete. Press any key to exit installation.
10. Proceed to Section 4.3 – "Registry Modification for Smart Card – CSP Association"

Note: If you rebuild the CSP at any time, you will need to run this batch file again

4.2 NIST-ESI CSP Installation Procedure

The procedure outlined in this subsection needs to be followed to use the NIST-ESI CSP without requiring any modification to the source code.

1. In the .\Binaries folder, run the file prepcsp.bat
2. Select option 1 "Install NIST CSP for PIV Card"
3. A message is displayed indicating installation is complete. Press any key to exit installation.
4. Proceed to Section 4.3 – "Registry Modification for Smart Card – CSP Association"

4.3 Registry Modification for Smart Card – CSP Association

Typically, smart cards contain a unique Answer-To-Reset (ATR) byte string that can be used to identify a smart card. In order to associate a smart card with a particular CSP, an association is made between the smart card's ATR and the CSP. The following subsections discuss the procedures for associating the PIV Card and PIV Reference Implementation with the NIST-ESI CSP.

4.3.1 Determine Card ATR

4.3.1.1 Determine ATR of PIV Card

The ATR of a PIV card can be determined by using the TestResMan[5] tool to establish a connection with the card and retrieve its ATR.

1. Launch TestResMan
2. Click Select Reader. A list of available smart card readers is displayed.
3. Select the smart card reader the card is inserted in and click OK
4. Click Card Connect. Share mode and protocol types are displayed.
5. Click OK to accept the default values and connect to the card
6. If a connection was successfully established with the card then no error messages should be displayed in the bottom left corner of the TestResMan dialog
7. Click Card State
8. The ATR is displayed in the Output as a string of hex characters

[5] TestResMan is available for download at http://www.scmmicro.com/support/pcs_downloads.php?lang=en.

9. Copy the ATR (TestResMan does not allow the Output's contents to be copied to the clipboard so it will have to be typed manually)
10. Click Card Disconnect
11. Select “Leave Card” and click OK. The card is now disconnected.
12. Click Cancel to close TestResMan

4.3.1.2 Determine ATR of PIV Reference Implementation

The ATR of the PIV Reference Implementation is set to 3B 90 96 40 0A. Hence, no additional steps are required to determine its ATR.

4.3.2 Associate ATR with CSP

The procedure outlined in this subsection needs to be followed to associate an ATR with the NIST-ESI CSP.

1. Open the registry editor and browse to the key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Calais\SmartCards
2. Right-click on the SmartCards key and select New->Key
3. Enter a name for the card (e.g. PIV Card)
4. Right-click on the newly created key and select New->Binary Value
5. Name this new binary value: **ATR**
6. Double-click ATR to edit its value and enter the ATR from section 4.3.1
7. Click OK to save
8. Add another binary value called: **ATRMask**
9. Double-click ATRMask to edit its value and enter FF for the number of bytes in the ATR above. For example, if the ATR is 17 bytes then enter: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF.
10. Click OK to save
11. Right-click on the key created in step 3 and select New->String Value
12. Name this new string value: **Crypto Provider**
13. Double-click Crypto Provider to edit its value and enter the following: NIST-ESI CSP
14. Click OK to save

Appendix A—Acronyms and Abbreviations

ADPU	Application Protocol Data Unit
API	Application Programming Interface
AS	Authentication Service
ATR	Answer To Reset
CA	Certification Authority
CSP	Cryptographic Service Provider
DLL	Dynamic Linked Library
FIPS	Federal Information Processing Standard
GINA	Graphical Identification and Authentication
HSPD	Homeland Security Presidential Directive
IETF	Internet Engineering Task Force
KDC	Key Distribution Center
LSA	Local Security Authority
MSCAPI	Microsoft Cryptographic Application Programming Interface
NIST	National Institute for Standards and Technology
NTLM	New Technology Loadable Module
PC/SC	Personal Computer/ Smart Card
PIN	Personal Identification Number
PIV	Personal Identity Verification
PKINIT	Public Key Cryptography for Initial Authentication in Kerberos
SAS	Secure Attention Sequence
SCCSP	Smart Card Cryptographic Service Provider
SP	Special Publication
TGS	Ticket Granting Service
TGT	Ticket Granting Ticket
USB	Universal Serial Bus

Appendix B—References

[1] FIPS Publication 201, *Personal Identity Verification (PIV) for Federal Employees and Contractors*. Available at http://csrc.nist.gov/publications/fips/fips201/FIPS-201-022505.pdf

[2] NIST, *PIV Middleware Reference Implementation User's Guide*, Version 1.1, June 27, 2005

[3] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnscard/html/smartcardcspcook.asp

[4] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthn/security/basic_authentication_concepts.asp

[5] NIST Special Publication 800-73, *Integrated Circuit Card for Personal Identity Verification*, NIST, February 2005. Available at http://csrc.nist.gov/publications/nistpubs/800-73/SP800-73-Final.pdf

[6] NIST, *PIV Windows Logon Reference Implementation: Best Practices and Troubleshooting*, June 2007