Technical Standard

CDSA/CSSM Authentication: Human Recognition Service (HRS) API Version 2

THE Open GROUP

[This page intentionally left blank]

Open Group Technical Standard

CDSA/CSSM Authentication:

Human Recognition Service (HRS) API, Version 2

The Open Group

Open Group Technical Standard

CDSA / CSSM Authentication: Human Recognition Service (HRS) API, Version 2

Document Number: C013

Published in the U.K. by The Open Group, June 2001.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

List of Figures

List of Tables

Preface

The Open Group

The Open Group is a vendor and technology-neutral consortium which ensures that multi-vendor information technology matches the demands and needs of customers. It develops and deploys frameworks, policies, best practices, standards, and conformance programs to pursue its vision: the concept of making all technology as open and accessible as using a telephone.

The mission of The Open Group is to deliver assurance of conformance to open systems standards through the testing and certification of suppliers' products.

The Open group is committed to delivering greater business efficiency and lowering the cost and risks associated with integrating new technology across the enterprise by bringing together buyers and suppliers of information systems.

Membership of The Open Group is distributed across the world, and it includes some of the world's largest IT buyers and vendors representing both government and commercial enterprises.

More information is available on The Open Group Web Site at *http://www.opengroup.org*.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available on The Open Group Web Site at *http://www.opengroup.org/pubs*.

- Product Standards

  A Product Standard is the name used by The Open Group for the documentation that records the precise conformance requirements (and other information) that a supplier's product must satisfy. Product Standards, published separately, refer to one or more Technical Standards.

  The ''X'' Device is used by suppliers to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the Open Brand Trademark License Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the supplier. The Open Group runs similar conformance schemes involving different trademarks and license agreements for other bodies.

- Technical Standards (formerly CAE Specifications)

  Open Group Technical Standards, along with standards from the formal standards bodies and other consortia, form the basis for our Product Standards (see above). The Technical Standards are intended to be used widely within the industry for product development and procurement purposes.

  Technical Standards are published as soon as they are developed, so enabling suppliers to proceed with development of conformant products without delay.

  Anyone developing products that implement a Technical Standard can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand.

- CAE Specifications

  CAE Specifications and Developers' Specifications published prior to January 1998 have the same status as Technical Standards (see above).

- Preliminary Specifications

  Preliminary Specifications have usually addressed an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. There is a strong preference to develop or adopt more stable specifications as Technical Standards.

- Consortium and Technology Specifications

  The Open Group has published specifications on behalf of industry consortia. For example, it published the NMF SPIRIT procurement specifications on behalf of the Network Management Forum (now TMF). It also published Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

In addition, The Open Group publishes Product Documentation. This includes product documentation—programmer's guides, user manuals, and so on—relating to the DCE, Motif, and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

Versions and Issues of Specifications

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.

- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on The Open Group Web Site at *http://www.opengroup.org/corrigenda*.

Ordering Information

Full catalog and ordering information on all Open Group publications is available on The Open Group Web Site at *http://www.opengroup.org/pubs*.

This Document

The CDSA/HRS (Common Data Security Architecture/Human Recognition Service) is a CSSM (Common Security Services Manager) EMM (Elective Module Manager).

Version 2 (this document) includes the changes approved in The Open Group Corrigendum U051, dated 16th March 2001, against Version 1 (Doc. No. C909).

This HRS API provides a high-level generic authentication model—one suited to use with any form of human authentication—for operation with CDSA. Particular emphasis has been put on designing it for performing authentication using biometric technology.

The development of this specification has passed through several organizational groups, but was integrated into a generic authentication API by the BioAPI Consortium; see **Acknowledgements** (on page xi).

This CDSA / HRS specification is based on BioAPI Version 1.0 8 (published March 30, 2000) and uses the EMM facilities provided in CSSM to provide a generic authentication service for CDSA. It covers the basic functions of *Enrollment*, *Verification*, and *Identification*, and includes a database interface to allow a biometric service provider (BSP) to manage the identification population for optimum performance. It also provides primitives which allow the application to manage the capture of samples on a client, and the functions of *Enrollment*, *Verification*, and *Identification*, on a server. It is designed to support multiple authentication methods, both singularly and when used in a combined or ''layered'' manner.

Audience

The HRS is designed for use by both application developers and biometric technology developers.

To make the integration of the technology as straightforward as possible (thus enhancing its commercial viability), the design approach has been to hide or encapsulate to the extent possible the complexities of the biometric technology. This approach also serves to extend the generality of the interface to address a larger set of potential biometric technologies and applications.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures, and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — Command operands, command option-arguments, or variable names; for example, substitutable argument prototypes
  - — Environment variables, which are also shown in capitals
  - — Utility names
  - — External variables, such as *errno*
  - — Functions; these are shown as follows: *name*( )
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header.
- Syntax, code examples, and user input in interactive examples are shown in `fixed width` font.

Trademarks

Acknowledgements

The Open Group acknowledges Intel's submission of its CDSA / HRS specification into The Open Group's ''Fast Track'' Company Review process, to become an Open Group Technical Standard.

The Open Group also acknowledges the work of the BioAPI Consortium in the development of this HRS specification. Intel were prime contributors to the BioAPI (Biological Authentication Application Programming Interface) Consortium. Special recognition is due to John Wilson (Intel) who chaired and did the technical editing for their Application Working Group, whose work resulted in a BioAPI specification.

The BioAPI Consortium published several revisions of their BioAPI specification, as their work evolved. Intel's CDSA / HRS specification is based on BioAPI Version 1.0 8, published March 30, 2000.

Information on the BioAPI Consortium is available from their web site at *http://www.bioapi.org*. The following list of BioAPI member organizations indicates the broad support for their BioAPI work on defining a common biologically-based authentication API and associated measurement devices and mechanisms:

Authentec
Barclays Bank
Biometric Identification, Inc.
BioNetrix
Business Integrated Technology Solutions (BITS)
Compaq *
Dialog Communications Systems AG
Digital Persona
Hewlett-Packard Company
IBM Corporation
Identix / Identicator
Image Computing Incorporated (ICI)
Infineon Technologies (formerly Siemens)
Integrated Visions, Inc.
Intel Corporation *
I / O Software, Inc.
IriScan *
ITT
J. Markowitz Consulting
Janus Associates
Kaiser Permanente
Keyware Technologies
Miros
Mytec Technologies *
National Biometrics Test Center
NIST *
NSA
OKI
Precise Biometrics
Recognition Systems
SAFLink *
Sagem-Morpho
Secugen
Sensar
Skytale
Startek
STMicroelectronics
Systemneeds, Inc.
Transaction Security
Transforming Technologies
TRW
UniSoft Corporation
Unisys *
Veridicom
Viatec Research
Visionics
Who?Vision

* Steering Committee Members

Referenced Documents

The following documents are referenced in this Technical Standard:

CDSA / CSSM

Technical Standard, May 2000, Common Security: CDSA and CSSM, Version 2 (ISBN: 1-85912-202-7, C914), published by The Open Group.

Chapter 1

Overview

1.1 Purpose

CDSA / HRS (Common Data Security Architecture / Human Recognition Service) is a CSSM (Common Security Services Manager) EMM (Elective Module Manager). It is intended to provide a high-level generic authentication model, suited to use for any form of human authentication. Particular emphasis has been made in the design on its suitability for authentication using biometric technology.

It covers the basic functions of *Enrollment*, *Verification*, and *Identification*, and includes a database interface to allow a biometric service provider (BSP) to manage the identification population for optimum performance.

It also provides primitives which allow the application to manage the capture of samples on a client, and the *Enrollment*, *Verification*, and *Identification*, on a server.

The HRS is designed for use by both application developers and biometric technology developers. To make the integration of the technology as straightforward and simple as possible (thus enhancing its commercial viability), the approach taken is to hide or encapsulate to the extent possible the complexities of the biometric technology. This approach also serves to extend the generality of the interface to address a larger set of potential biometric technologies and applications.

This specification is designed to support multiple authentication methods, both singularly and when used in a combined or ‘‘layered’’ manner.

1.2 Biometric Technology

The basic model is the same for all types of biometric technology. First, the initial registration ‘‘template’’ of the user has to be constructed. This is done by collecting a number of samples through whatever sensor is being used. Salient features are extracted from the samples, and the results combined into the template. The construction of this initial template is called *Enrollment*. The algorithms used to construct a template are usually proprietary. This initial template is then stored by the application, and essentially takes the place of a password.

Thereafter, whenever the user needs to be authenticated, live samples are captured from the device, processed into a usable form, and matched against the template that was enrolled earlier. This form of biometric authentication is called *Verification*, since it verifies that a user is who they say they are (that is, verifies a particular asserted identity).

Biometric technology, however, allows a new form of authentication called *Identification*. In this form, a user does not have to assert an identity. Instead, the biometric service provider compares the processed samples of the user against a specified population of templates, and decides which ones match most closely. Depending on the match probabilities, the user’s identity could be concluded to be that corresponding to the template with the closest match.

Figure 1-1 shows some possible implementation strategies. The various steps in the verification and identification operations are shown in the box labeled ‘‘Biometric Service Provider’’. The stages identified above the box refer to the primitive functions of the API:

- Capture
- Process
- Match

and it shows that a BSP has degrees of freedom in the placement of function in these primitives. There is a degree of freedom not shown in Figure 1-1; the manufacturer is free to put most, if not all, of the BSP functionality in the sensing device itself. In fact, if the device contains a Biometric Identification Record (BIR) database, all functions may be performed in the device.

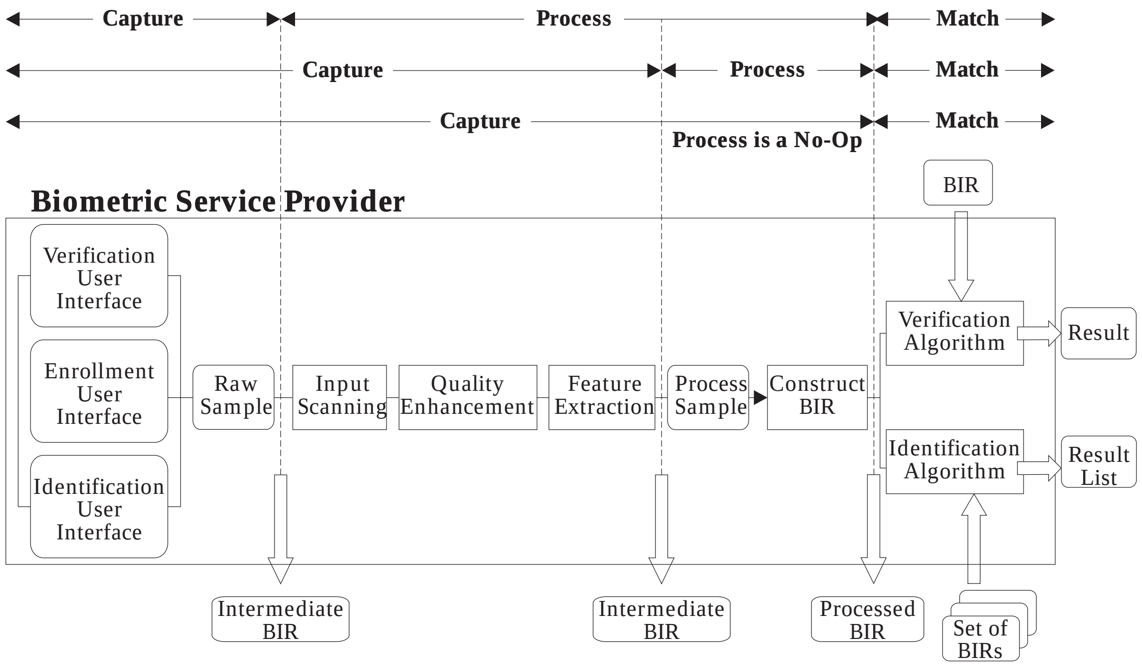

Figure 1-1 Possible Implementation Strategies

1.3 BIRs and Templates

This document uses the term *template* to refer to the biometric enrollment data for a user. The template must be matched (within a specified tolerance) by sample(s) taken from the user, in order for the user to be authenticated.

The term *biometric identification record* (BIR) refers to any biometric data that is returned to the application, including raw data, intermediate data, processed sample(s) ready for verification or identification, as well as enrollment data. Typically, the only data stored persistently by the application is the BIR generated for enrollment (that is, the template). The structure of a BIR is shown in Figure 1-2.

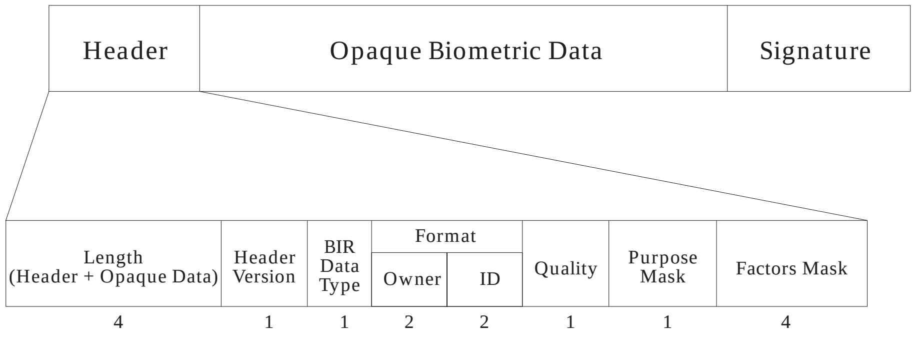

Figure 1-2 Biometric Identification Record (BIR)

The format of the *Opaque Biometric Data* is indicated by the *Format* field of the *Header*. This may be a standard or proprietary format. The *Opaque Data* may be encrypted.

Values of *Format Owner* are assigned and registered by the International Biometric Industry Association (IBIA), which ensures uniqueness of these values. Registered format owners then create one or more *FormatIDs* (either published or proprietary), corresponding to a defined format for the subsequent opaque biometric data, which may optionally also be registered with the IBIA. Organizations wishing to register as a biometric data format owner can do so for a nominal fee by contacting the IBIA as follows:

International Biometric Industry Association (IBIA)
601 Thirteenth Street, NW, Suite 370 South, Washington DC. 20005, USA
Tel: +1 202-783-7272
Fax: +1 202-783-4345
http://www.ibia.org/formats.htm

The BIR definition is compliant with NISTIR 6529, Common Biometric Exchange File Format (CBEFF), of which it is one of the CBEFF Patron Formats.

The *Signature* is optional. When present, it is calculated on the Header + Biometric Data. For standardized BIR formats, the signature will take a standard form (to be determined when the format is standardized). For proprietary BIR formats (all that exists at the present time), the signature can take any form that suits the BSP. For this reason, there is no C program structure definition of the signature.

The BIR *Data Type* indicates whether the BIR is signed and/or encrypted.

When a service provider creates a new BIR, it returns a handle to it. Most local operations can be performed without moving the BIR out of the service provider. [BIRs can be quite large, so this is a performance advantage.] However, if the application needs to manage the BIR (either store it in an application database, or send it to a server for verification/identification), it can acquire the BIR using the handle.

Whenever an application needs to provide a BIR as input, it can be done in one of three ways:

1. By reference to its handle
2. By reference to its key value in an open database managed by the BSP
3. By supplying the BIR itself

1.4 API Model

There are three principal high-level abstraction functions in the API:

1. *Enroll*

   Samples are captured from a device, processed into a usable form from which a template is constructed, and returned to the application.

2. *Verify*

   One or more samples are captured, processed into a usable form, and then matched against an input template. The results of the comparison are returned.

3. *Identify*

   One or more samples are captured, processed into a usable form, and matched against a set of templates. A list is returned showing how close the samples compare against the top candidates in the set.

However, as Figure 1-1 (on page 2) shows, the processing of the biometric data from the capture of raw samples to the matching against a template may be accomplished in many stages, with much CPU-intensive processing. The API has been defined to allow the biometric developer the maximum freedom in the placement of the processing involved, and allows the processing to be shared between the client machine (which has the biometric device attached) and a server machine. It also allows for self-contained devices, in which all the biometric processing can be done internally. Client/server support by BSPs is optional.

There are several good reasons why processing and matching may take place on a server, including:

- The algorithms will execute in a more secure environment.
- The Client PC may not have sufficient power to run the algorithms well.
- The user database (and the resources that it is protecting) may be on a server.
- Identification over large populations can only reasonably be done on a server.

Two methods are provided to support client/server processing:

1. **Using Primitive Functions**

   There are four **primitive** functions in the API which, when used in sequence on client and server, can accomplish the same result as the high-level abstractions:

   Capture — *Capture* is always executed on the client machine; attempting to execute *Capture* on a machine without a biometric device will return *function not supported*. One or more samples are acquired (either for *Enrollment*, *Verification*, or *Identification*). The *Capture* function is allowed to perform as much processing on the sample(s) as it sees fit, and may, in fact, for verification or identification, complete the construction of the BIR. If processing is incomplete, *Capture* returns an ‘‘intermediate’’ BIR; indicating that the *Process* function needs to be called. If processing is complete, *Capture* returns a ‘‘processed’’ BIR; indicating that the *Process* function does not need to be called. The application specifies the purpose for which the samples are intended, giving the BSP the opportunity to do special processing. This purpose is recorded in the header of the constructed BIR.

Process	The ‘‘processing algorithms’’ must be available on the server, but may also be available on the client. The *Process* function is intended to provide the processing of samples necessary for the purpose of verification or identification (not enrollment). It always takes an ‘‘intermediate’’ BIR as input, and may complete the processing of the biometric data into ‘‘final’’ form suitable for its intended purpose. On the client, if it completes the processing, it returns a ‘‘processed’’ BIR; otherwise, it returns an ‘‘intermediate’’ BIR, indicating that *Process* needs to be called on the server. On the server, it will always complete processing, and always return a ‘‘processed’’ BIR. The application can always choose to defer processing to the server machine, but may try to save bandwidth (and server horsepower) by calling *Process* on the client.[1]
Match	Performs the actual comparison between the ‘‘processed’’ BIR and one template (*VerifyMatch*), or between the ‘‘processed’’ BIR and a set of templates (*IdentifyMatch*). The support for *IdentifyMatch* is optional, but the supported *Match* functions are always available on the server, and may be available on the client.
CreateTemplate	*CreateTemplate* is provided to perform the processing of samples for the construction of an enrollment template. *CreateTemplate* always takes an ‘‘intermediate’’ BIR as input, and constructs a template (that is, a ‘‘processed’’ BIR with the recorded purpose of either *enroll_verify* and/or *enroll_identify*). Optionally, *CreateTemplate* can take an old template and create a new template which is the adaptation of the old template using the new biometric samples in the ‘‘intermediate’’ BIR. The BSP may optionally allow the application to provide a ‘‘payload’’ to wrap inside the new template; see Section 1.6 (on page 8).

1. ‘‘Processed’’ BIRs are always smaller than ‘‘intermediate’’" BIRs; by how much is technology-dependent, and also dependent on how much processing has already been done by *Capture*.

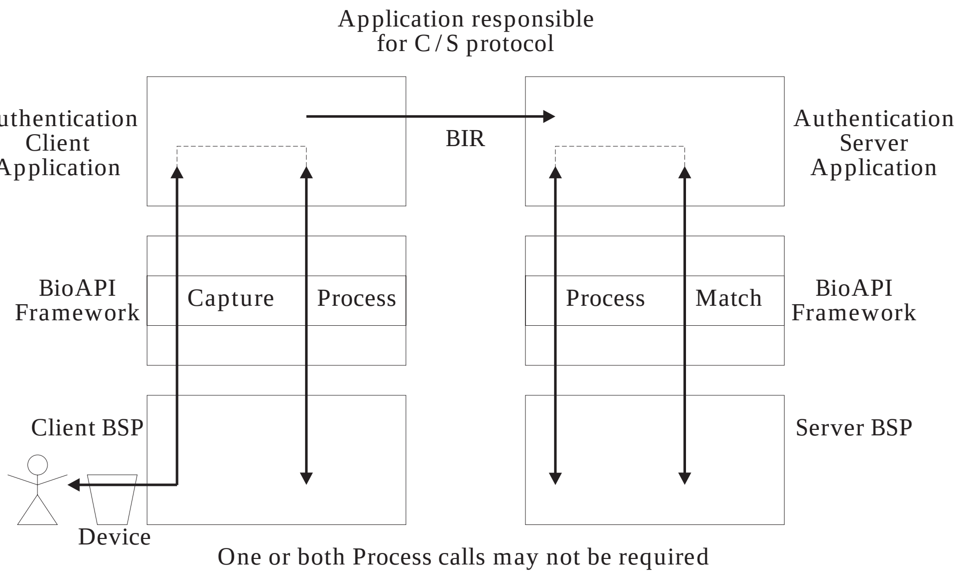
Application responsible for C / S protocol
Authentication Client Application
BIR
Authentication Server Application
BioAPI Framework
Capture
Process
Process
Match
BioAPI Framework
Client BSP
Server BSP
Device
One or both Process calls may not be required

Figure 1-3 Client / Server Implementation Using Primitive Functions

2. **Using a Streaming Callback**

   The application (on both the client and the server) is responsible for providing a streaming interface for the BSP to use to communicate the samples and return the results. In this case, the application does not need to use the *Capture*, *Process* and *Match* primitives. The *Verify*, *Identify*, and *Enroll*, functions use the streaming interface to split the BSP function between client and server. These functions may be driven from either the client or the server. In either case, if there are Graphical User Interface (GUI) callbacks set, the client BSP will call them at the appropriate times to allow the client application to control the look and feel of the user interface.

   - The client / server application decides whether the authentication should be driven by the client or the server component. The driving component first sets a *Streaming Callback* interface for the BSP. This not only tells the BSP that it is going to operate in client / server mode, but also provides the interface that it will use to initiate communication with its partner BSP.
   - The application calls the appropriate high-level function, and the BSP calls the *Streaming Callback* to initiate the BSP-to-BSP protocol. (The protocol is the concern of the BSP implementer, but it will likely start with mutual authentication and key agreement).
   - The *Streaming Callback* is only used by the driving BSP. Whenever it is in control, and has a message to deliver to its partner BSP, it calls the *Streaming Callback* interface to send the message, and it receives an answer on return from the callback.
   - The *StreamInputOutput* function is used by the partner application to deliver messages to the partner BSP, and to obtain a return message to send to the driving BSP. The driving application delivers the return message by returning from the *Streaming Callback*.

Note: A client BSP that is servicing a self-contained device may ignore the *Streaming Callback* interface, and perform the requested function locally. A server BSP that is servicing a self-contained device will use the *Streaming Callback* to request that the client BSP proceeds to perform the requested function.

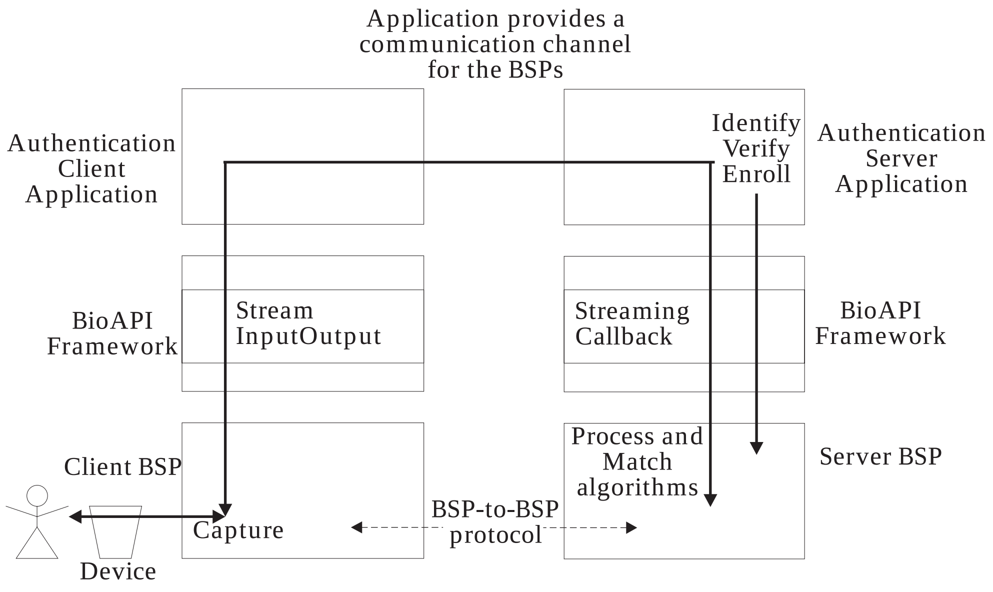
Application provides a communication channel for the BSPs
Authentication Client Application
Identify Verify Enroll
Authentication Server Application
BioAPI Framework
Stream InputOutput
Streaming Callback
BioAPI Framework
Client BSP
Process and Match algorithms
Server BSP
Capture
BSP-to-BSP protocol
Device

Figure 1-4 Client / Server Impl. Using Streaming Callback: Server-Initiated Operation

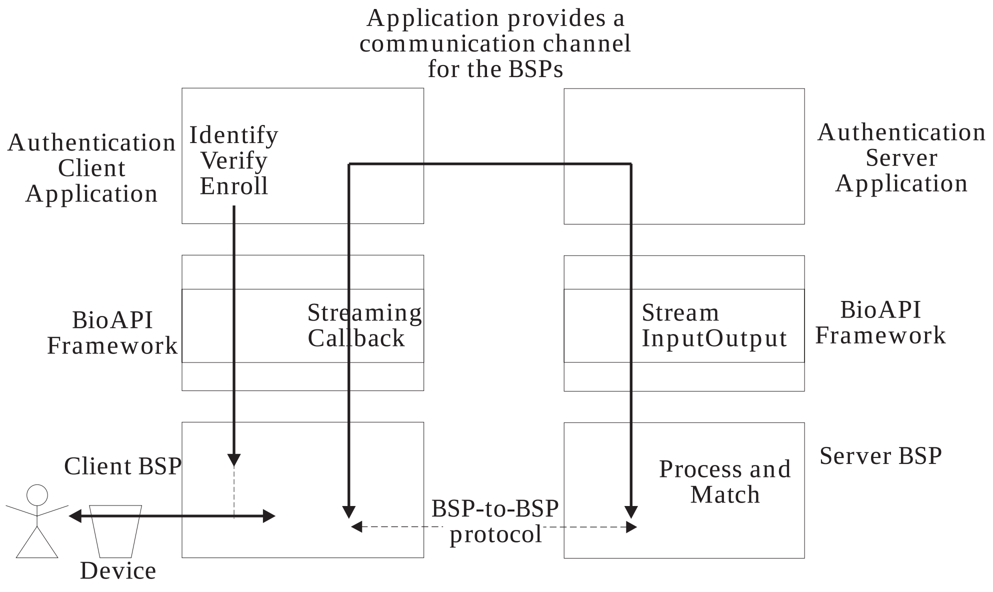

Figure 1-5 Client / Server Impl. Using Streaming Callback: Client-Initiated Operation

1.5 FAR and FRR

Raw biometric samples are the complex analog data streams produced by sensing devices. No two samples from a user are likely to be identical. Templates are the digital result of processing and compressing these samples; they are not precise representations of the user. Therefore, the results of any matching of samples against a stored template can only be expressed in terms of probability.

There are two possible criteria for the results of a match:

1. False Accept Rate (FAR)
2. False Reject Rate (FRR)

FAR is the probability that samples falsely match the presented template, whereas FRR is the probability that the samples are falsely rejected (that is, should match, but do not). Depending on the circumstances, the application may be more interested in one than the other.

The HRS functions allow the application to request a match threshold in terms of maximum FAR value (that is, a limit on the probability of a false match) and an optional maximum FRR value. If both are provided, the application must tell the BSP which one should take precedence.

The principal result returned is the actual FAR value achieved by the match (that is, the score). A BSP may optionally return the actual FRR value achieved. For *Identify* and *IdentifyMatch*, these results are contained in the *Candidate* array.

The returning of scores to an application can be a security weakness if appropriate steps are not taken. This is because a ‘‘rogue’’ application can mount a ‘‘hillclimbing’’ attack by sequentially randomly modifying a sample and retaining only the changes that produce an increase in the returned score. In this way a synthetic image can be created to fool the biometric system. However, allowing only discrete increments of score (FAR) to be returned to the application eliminates this method of attack. The level of quantization required to neutralize this attack is dependent on the type of biometric and the algorithms used.

Use of FAR/FRR values to represent match scores is done to allow a degree of normalization and comparison between differing technologies, and to allow a commonly understood means of setting thresholds and interpreting results. It is not intended to imply strict performance measurement (that is, an absolute measure of FAR for a specific individual matching instance). Furthermore, the BSP vendor is responsible for accurately mapping internal scoring structure to the FAR values.

1.6 Payloads

No two biometric samples from a user are likely to be identical. For this reason, it is not possible to directly use biometric samples as cryptographic keys. The HRS, however, allows a template to be closely bound to a cryptographic key, which could be released upon successful verification.

In the *Enroll* and *CreateTemplate* functions, the application may present a ‘‘payload’’ to be carried in the opaque data of the BIR that is being constructed. This payload is essentially wrapped inside the biometric data by the BSP. It is released to the application on successful verification of the template. The BSP may have a policy of only releasing the ‘‘payload’’ if the actual FAR achieved is below a certain threshold (this threshold being recorded in the BSP’s registry entry).

The ‘‘payload’’ can be any data that is useful to the application; it does not have to be cryptographic; and even in a cryptographic application it could be either a key label or a wrapped key.

1.7 BIR Databases

The HRS does not manage user databases; only applications do that. In most cases, the user database may already exist (for example, a database of bank accounts, the user registry of people who belong to a network domain, or those authorized to access a web server), and the biometric application is simply associating a biometric template with each user in the database, in addition to (or as a substitute for) a password. It is important that the application does maintain control over who can access this database.

The HRS allows a BSP to manage a database of BIRs for two reasons:

1. To optimize the performance of the *Identification* operation over large populations
2. To provide access to the BIRs that may be stored on a self-contained sensing device

It is the responsibility of the application to make any necessary association between the BSP's database(s) and the user database(s). To assist in this, each entry in a BSP database has a GUID associated with it. For security reasons, entries in a BSP database cannot be modified; only created and deleted. New entries get new GUIDs.

Not all service providers support identification, and not all those need to support a database interface. If the identification population is sufficiently small, it can be handled by passing an array of BIRs across the interface.

Databases can be created by name, and a service provider may have a default database. If a service provider supports a device that can store BIRs, then that device should be the default database. The default database is always open when the BSP is attached, and the handle to the open default database is always –1.

1.8 User Interface Considerations

The user interface for passwords and PINs is quite straightforward, but for biometric technology it can be quite complex and very much technology-dependent, requiring multiple implementation-dependent interactions with the user. Some biometric technologies present streams of data to the user (face and voice, for example), while some require the user to validate each sample taken (face, voice, and signature, for example). During enrollment, some technologies verify each sample taken against the previous samples. The number of samples taken for a particular purpose may vary from technology to technology, and finally, the user interface is generally different for enrollment than for verification and identification.

Most biometric service providers come with a built-in user interface, and this may often be sufficient for most purposes. The API, however, allows the application to control the ‘‘look-and-feel’’ of this user interface by allowing the application to provide callbacks for the service provider to use to present and gather samples.

One of the callbacks is used to present and gather samples and to indicate state changes to the application. All service providers implementing the *Application Controlled GUI* option must support this callback, though the state machines may vary considerably. The other GUI callback is used to present streaming data to the user, in the form of a series of bitmaps. This callback is optional, and the service provider indicates in its registry entry whether one is required. This entry also indicates whether the user must validate samples, and whether samples are verified.

The service provider is in control of the user interface state machine, and calls the state callback whenever there is a state change event. These state changes may be the completion of a sample, progress on a sample, or the need to present the user with a message. On return, the application can present the service provider with a response from the user; cancel, continue, valid sample, invalid sample, and so on.

If the service provider needs to present a sample stream to the user, it calls the streaming callback in parallel to the state change events. This callback will require multi-threading in both the service provider and the application.

Chapter 2

API Definition

2.1 Data Structures

2.1.1 CSSM_MODULE_EVENT

The following two events are added for HRS:

```
#define CSSM_NOTIFY_SOURCE_PRESENT        (4)
#define CSSM_NOTIFY_SOURCE_REMOVED        (5)
```

2.1.2 CSSM_MODULE_EVENT_MASK

This enumeration defines a mask with bit positions for event type. The mask is used to enable / disable events, and to indicate what events are supported.

```
typedef uint32 CSSM_MODULE_EVENT_MASK;
#define CSSM_NOTIFY_INSERT_BIT            (0x0001)
#define CSSM_NOTIFY_REMOVE_BIT            (0x0002)
#define CSSM_NOTIFY_FAULT_BIT             (0x0004)
#define CSSM_NOTIFY_SOURCE_PRESENT_BIT    (0x0008)
#define CSSM_NOTIFY_SOURCE_REMOVED_BIT    (0x0010)
```

2.1.3 CSSM_HRS_BIR

A container for biometric data. A BIR may contain raw sample data, partially processed (intermediate) data, or completely processed data. It may be used to enroll a user (thus being stored persistently), or may be used to verify or identify a user (thus being used transiently).

The opaque biometric data is of variable length, and may be followed by a signature. The signature itself may not be a fixed length, depending on which signature standard is employed. The signature is calculated on the combined *Header* and *BiometricData*.

```
typedef struct cssm_hrs_bir {
    CSSM_HRS_BIR_HEADER  Header;
    CSSM_HRS_BIR_BIOMETRIC_DATA_PTR BiometricData;
        /* Length indicated in header. */
    CSSM_DATA_PTR Signature;
        /* NULL if no signature; length is inherent in this type. */
} CSSM_HRS_BIR, *CSSM_HRS_BIR_PTR;
```

2.1.4 CSSM_HRS_BIR_ARRAY_POPULATION

An array of BIRs, generally used during identification operations (as input to *Identify* or *Identify_Match*).

```
typedef struct cssm_hrs_bir_array_population {
    uint32 NumberOfMembers;
    CSSM_HRS_BIR_PTR *Members;
        /* A pointer to an array of BIR pointers. */
} CSSM_HRS_BIR_ARRAY_POPULATION, *CSSM_HRS_BIR_ARRAY_POPULATION_PTR;
```

2.1.5 CSSM_HRS_BIR_AUTH_FACTORS

A mask that describes the set of authentication factors supported by an authentication service.

```
typedef uint32 CSSM_HRS_BIR_AUTH_FACTORS;
#define CSSM_HRS_FACTOR_MULTIPLE              (0x00000001)
#define CSSM_HRS_FACTOR_FACIAL_FEATURES       (0x00000002)
#define CSSM_HRS_FACTOR_VOICE                 (0x00000004)
#define CSSM_HRS_FACTOR_FINGERPRINT           (0x00000008)
#define CSSM_HRS_FACTOR_IRIS                  (0x00000010)
#define CSSM_HRS_FACTOR_RETINA                (0x00000020)
#define CSSM_HRS_FACTOR_HAND_GEOMETRY         (0x00000040)
#define CSSM_HRS_FACTOR_SIGNATURE_DYNAMICS    (0x00000080)
#define CSSM_HRS_FACTOR_KEYSTOKE_DYNAMICS     (0x00000100)
#define CSSM_HRS_FACTOR_LIP_MOVEMENT          (0x00000200)
#define CSSM_HRS_FACTOR_THERMAL_FACE_IMAGE    (0x00000400)
#define CSSM_HRS_FACTOR_THERMAL_HAND_IMAGE    (0x00000800)
#define CSSM_HRS_FACTOR_GAIT                  (0x00001000)
#define CSSM_HRS_FACTOR_PASSWORD              (0x80000000)
```

Note: All integer values in the BIR header are little-endian.

2.1.6 CSSM_HRS_BIR_BIOMETRIC_DATA

This comprises the *opaque* data block within a BIR containing the biometric sample(s) or template(s).

```
typedef uint8 CSSM_HRS_BIR_BIOMETRIC_DATA, \
    *CSSM_HRS_BIR_BIOMETRIC_DATA_PTR;
```

Note: The format of this data is specified by the format field (CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT) in the BIR header; see Section 1.3 (on page 2).

2.1.7 CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT

Defines the format of the data contained within the *opaque* data block, CSSM_HRS_BIR_BIOMETRIC_DATA.

```
typedef struct cssm_hrs_bir_biometric_data_format {
    uint16 FormatOwner;
    uint16 FormatID;
} CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT, \
    *CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT_PTR;
```

Notes:

1. *FormatOwner* values are assigned and registered by the International Biometric Industry Association (IBIA). *FormatID* is assigned by the Format Owner and may optionally be registered by the IBIA. Contact information for the IBIA is located in Section 1.3 (on page 2).
2. All integer values in the BIR header are little-endian.

2.1.8 CSSM_HRS_BIR_DATA_TYPE

Mask bits that may be OR'ed together to indicate the type of opaque data in the BIR (Raw OR Intermediate, OR Processed, OR Encrypted, OR Signed.)

```
typedef uint8 CSSM_HRS_BIR_DATA_TYPE;

#define CSSM_HRS_BIR_DATA_TYPE_RAW              (0x01)
#define CSSM_HRS_BIR_DATA_TYPE_INTERMEDIATE     (0x02)
#define CSSM_HRS_BIR_DATA_TYPE_PROCESSED        (0x04)
#define CSSM_HRS_BIR_DATA_TYPE_ENCRYPTED        (0x10)
#define CSSM_HRS_BIR_DATA_TYPE_SIGNED           (0x20)
```

Note: All integer values in the BIR header are little-endian.

2.1.9 CSSM_HRS_BIR_HANDLE

A handle to refer to BIR data that exists in the service provider.

```
typedef  sint32 CSSM_HRS_BIR_HANDLE, *CSSM_HRS_BIR_HANDLE_PTR;
#define CSSM_HRS_INVALID_BIR_HANDLE        (-1)
#define CSSM_HRS_UNSUPPORTED_BIR_HANDLE    (-2)
```

Note: All integer values in the BIR header are little-endian.

2.1.10 CSSM_HRS_BIR_HEADER

```
typedef struct cssm_hrs_bir_header {
    uint32 Length;    /* Length of Header + Opaque Data */
    CSSM_HRS_BIR_VERSION HeaderVersion;
    CSSM_HRS_BIR_DATA_TYPE Type;
    CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT Format;
    CSSM_HRS_QUALITY Quality;
    CSSM_HRS_BIR_PURPOSE Purpose;
    CSSM_HRS_BIR_AUTH_FACTORS FactorsMask;
} CSSM_HRS_BIR_HEADER, *CSSM_HRS_BIR_HEADER_PTR;
```

2.1.11 CSSM_HRS_BIR_PURPOSE

A value which defines the purpose(s) or use(s) for which the BIR is intended (when used as an input) or suitable (when used as an output or within the BIR header).

```
typedef uint8 CSSM_HRS_BIR_PURPOSE;

#define CSSM_HRS_PURPOSE_VERIFY                              (1)
#define CSSM_HRS_PURPOSE_IDENTIFY                            (2)
#define CSSM_HRS_PURPOSE_ENROLL                              (3)
#define CSSM_HRS_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY        (4)
#define CSSM_HRS_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY      (5)
#define CSSM_HRS_PURPOSE_AUDIT                               (6)
```

Notes:

1. All integer values in the BIR header are little-endian.
2. The Purpose value is utilized in two ways.

   First, it is used as an input parameter to allow the application to indicate to the BSP the purpose for which the resulting data is intended, thus allowing the BSP to perform the appropriate capturing and/or processing to create the proper BIR for this purpose.

   Second, it is used within the BIR header to indicate to the application (or to the BSP during subsequent operations) what purposes the BIR is suitable for.

   For example, some BSPs use different BIR formats depending on whether the data is to be used for verification or identification, the latter generally including additional information to enhance speed or accuracy. Similarly, many BSPs use different data formats depending on whether the data is to be used as a sample for immediate verification or as a reference template for future matching (that is, enrollment.)

   Restrictions on the use of BIR data of a particular purpose include:

   - All purposes are valid in the BIR header.
   - Purposes of *Verify* and *Identify* are only valid as input to the *Capture* function.
   - Purposes of *Enroll*, *Enroll_for_Verification_Only*, and *Enroll_for_Identification_Only* are only valid as input to the *Capture*, *Enroll*, and *Import* functions.
   - The *Audit* purpose is not valid as input to any function, but is only used in the BIR header.
   - The *Process* and *Create_Template* functions do not have *Purpose* as an input parameter, but read the *Purpose* field from the BIR header of the input *Captured_BIR*.
   - The *Process* function may accept as input any intermediate BIR with a *Purpose* including *Verify* or *Identify*, and will output only BIRs with a *Purpose* of *Verify* and/or *Identify*.
   - The *Create_Template* function may accept as input any intermediate BIR with a *Purpose* including *Enroll*, *Enroll_for_Verification_Only*, and/or *Enroll_for_Identification*, and will output only BIRs with a *Purpose* including that of the input BIR.
   - If a BIR is suitable for enrollment for either subsequent verification or identification, then the *Enroll Purpose* is *to* be *used* in

2.1.12 CSSM_HRS_BIR_VERSION

This data type is used to represent the version of a BIR header. The first version has a value of 1.

```
typedef  uint8  CSSM_HRS_BIR_VERSION, *CSSM_HRS_BIR_VERSION_PTR;
```

Note: All integer values in the BIR header are little-endian.

2.1.13 CSSM_HRS_CANDIDATE

One of a set of candidates returned by *Identify* or *IdentifyMatch*, indicating a successful match.

```
typedef struct cssm_hrs_candidate {
    CSSM_HRS_IDENTIFY_POPULATION_TYPE Type;
    union {
        CSSM_GUID_PTR BIRInDataBase;
        uint32 *BIRInArray;
    } BIR;
    CSSM_HRS_FAR  FARAchieved;
    CSSM_HRS_FRR  FRRAchieved;
} CSSM_HRS_CANDIDATE, *CSSM_HRS_CANDIDATE_PTR;
```

2.1.14 CSSM_HRS_CANDIDATE_ARRAY

An array of candidates returned by *Identify* or *IdentifyMatch*.

```
typedef CSSM_HRS_CANDIDATE_PTR CSSM_HRS_CANDIDATE_ARRAY, \
    *CSSM_HRS_CANDIDATE_ARRAY_PTR;
```

2.1.15 CSSM_HRS_DB_ACCESS_TYPE

This bitmask describes a user's desired level of access to a database. The BSP may use the mask to determine what lock to obtain on a database.

```
typedef uint32 CSSM_HRS_DB_ACCESS_TYPE, *CSSM_HRS_DB_ACCESS_TYPE_PTR;

#define CSSM_HRS_DB_ACCESS_READ    (0x1)
#define CSSM_HRS_DB_ACCESS_WRITE   (0x2)
```

2.1.16 CSSM_HRS_DB_CURSOR

A handle to a record in an open BIR database. The internal state for the handle includes the open database handle and also the position of a record in that open database. All cursors to an open database are freed when the database is closed.

```
typedef uint32 CSSM_HRS_DB_CURSOR, *CSSM_HRS_DB_CURSOR_PTR;
```

2.1.17 CSSM_HRS_DB_HANDLE

A handle to an open BIR database.

```
typedef sint32 CSSM_HRS_DB_HANDLE, *CSSM_HRS_DB_HANDLE_PTR;
#define CSSM_HRS_DB_DEFAULT_DB_HANDLE    (-1)
#define CSSM_HRS_DB_INVALID_HANDLE       (-2)
```

2.1.18 CSSM_HRS_DBBIR_ID

A structure providing the handle to a database managed by the BSP, and the ID of a BIR in that database.

```
typedef struct cssm_hrs_dbbir_id {
    CSSM_HRS_DB_HANDLE DbHandle;
    CSSM_GUID KeyValue;
} CSSM_HRS_DBBIR_ID, *CSSM_HRS_DBBIR_ID_PTR;
```

2.1.19 CSSM_HRS_FAR

A 32-bit integer value (N) that indicates a probable False Accept Rate of N / (231–1). The larger the value, the worse the result.

```
typedef sint32 CSSM_HRS_FAR, *CSSM_HRS_FAR_PTR;
#define CSSM_HRS_FAR_NOT_SET    (-1)
```

FAR is used within BioAPI as a means of setting thresholds and returning scores; see Section 1.5 (on page 8).

2.1.20 CSSM_HRS_FRR

A 32-bit integer value (N) that indicates a probable False Reject Rate of N / (231–1). The larger the value, the worse the result.

```
typedef sint32  CSSM_HRS_FRR, *CSSM_HRS_FRR_PTR;
#define CSSM_HRS_FRR_NOT_SET          (-1)
#define CSSM_HRS_FRR_NOT_SUPPORTED    (-2)
```

FRR is used within CDSA / HRS as an optional / alternate means of setting thresholds and returning scores; see Section 1.5 (on page 8).

2.1.21 CSSM_HRS_GUI_BITMAP

```
typedef struct cssm_hrs_gui_bitmap {
    uint32 Width;
    uint32 Height;
    CSSM_DATA_PTR Bitmap;
} CSSM_HRS_GUI_BITMAP, *CSSM_HRS_GUI_BITMAP_PTR;
```

2.1.22 CSSM_HRS_GUI_MESSAGE

```
typedef uint32 CSSM_HRS_GUI_MESSAGE, *CSSM_HRS_GUI_MESSAGE_PTR;
```

2.1.23 CSSM_HRS_GUI_PROGRESS

```
typedef uint8 CSSM_HRS_GUI_PROGRESS, *CSSM_HRS_GUI_PROGRESS_PTR;
```

2.1.24 CSSM_HRS_GUI_RESPONSE

```
typedef uint8 CSSM_HRS_GUI_RESPONSE;
#define CSSM_HRS_CAPTURE_SAMPLE    (1)
#define CSSM_HRS_CANCEL            (2)
#define CSSM_HRS_CONTINUE          (3)
#define CSSM_HRS_VALID_SAMPLE      (4)
#define CSSM_HRS_INVALID_SAMPLE    (5)
```

2.1.25 CSSM_HRS_GUI_STATE

A mask that indicates GUI state, and also what other parameter values are provided in the GUI State Callback.

```
typedef uint32 CSSM_HRS_GUI_STATE;

#define CSSM_HRS_SAMPLE_AVAILABLE     (0x0001)
#define CSSM_HRS_MESSAGE_PROVIDED     (0x0002)
#define CSSM_HRS_PROGRESS_PROVIDED    (0x0004)
```

2.1.26 CSSM_HRS_GUI_STATE_CALLBACK

A *Callback* function that an application supplies to allow the service provider to indicate GUI state information to the application, and to receive responses back.

```
typedef CSSM_RETURN (CSSMAPI *CSSM_HRS_GUI_STATE_CALLBACK)
    (void *GuiStateCallbackCtx,
    CSSM_HRS_GUI_STATE GuiState,
    CSSM_HRS_GUI_RESPONSE Response,
    CSSM_HRS_GUI_MESSAGE Message,
    CSSM_HRS_GUI_PROGRESS Progress,
    CSSM_HRS_GUI_BITMAP_PTR SampleBuffer);
```

Parameters

GuiStateCallbackCtx (input)
A generic pointer to context information that was provided by the original requester and is being returned to its originator.

GuiState (input)
An indication of the current state of the service provider with respect to the GUI, plus an indication of what others parameters are available.

Response (output)
The response from the application back to the service provider on return from the *Callback*.

Message (input/optional)
The number of a message to display to the user. Message numbers are service provider-dependent. *GuiState* indicates whether a message is provided; if not, the parameter is NULL.

Progress (input/optional)
A value that indicates (as a percentage) the amount of progress in the development of a Sample/BIR. The value may be used to display a progress bar. Not all service providers support a progress indication. *GuiState* indicates whether a sample progress value is provided in the call; if not, the parameter is NULL.

SampleBuffer (input/optional)
The current sample buffer for the application to display. *GuiState* indicates whether a sample buffer is provided; if not, the parameter is NULL.

2.1.27 CSSM_HRS_GUI_STREAMING_CALLBACK

A *Callback* function that an application supplies to allow the service provider to stream data in the form of a sequence of bitmaps.

```
typedef CSSM_RETURN (CSSMAPI *CSSM_HRS_GUI_STREAMING_CALLBACK)
    (void *GuiStreamingCallbackCtx,
    CSSM_HRS_GUI_BITMAP_PTR Bitmap);
```

Parameters

GuiStreamingCallbackCtx (input)
A generic pointer to context information that was provided by the original requester and is being returned to its originator.

Bitmap (input)
A pointer to the bitmap to be displayed.

2.1.28 CSSM_HRS_HANDLE

```
typedef CSSM_MODULE_HANDLE CSSM_HRS_HANDLE;  /* HRS Module Handle */
```

This data structure represents the HRS module handle. The handle value is a unique pairing between an HRS module and an application that has attached that module. HRS handles can be returned to an application as a result of the *CSSM_ModuleAttach*( ) function.

2.1.29 CSSM_HRS_IDENTIFY_POPULATION

A structure used to identify the set of BIRs to be used as input to an *Identify* or *Identify_Match* operation.

```
typedef struct cssm_hrs_identify_population {
    CSSM_HRS_IDENTIFY_POPULATION_TYPE Type;
    union {
        CSSM_HRS_DB_HANDLE_PTR  BIRDataBase;
        CSSM_HRS_BIR_ARRAY_POPULATION_PTR  BIRArray;
    } BIRs;
} CSSM_HRS_IDENTIFY_POPULATION, *CSSM_HRS_IDENTIFY_POPULATION_PTR;
```

2.1.30 CSSM_HRS_IDENTIFY_POPULATION_TYPE

A value indicating the method of BIR input to an *Identify* or *Identify_Match* operation, whether via a passed-in array or a pointer to a database.

```
typedef uint8 CSSM_HRS_IDENTIFY_POPULATION_TYPE;

#define CSSM_HRS_DB_TYPE       (1)
#define CSSM_HRS_ARRAY_TYPE    (2)
```

2.1.31 CSSM_HRS_INPUT_BIR

A structure used to input a BIR to the API. Such input can be in one of three forms:

1. A BIR Handle
2. A key to a BIR in a database managed by the BSP. If the *DbHandle* is zero, the default database is assumed. (A *DbHandle* is returned when a database is opened).
3. An actual BIR

```
typedef struct cssm_hrs_input_bir {
    CSSM_HRS_INPUT_BIR_FORM Form;
    union {
        CSSM_HRS_DBBIR_ID_PTR  BIRinDb;
        CSSM_HRS_BIR_HANDLE_PTR  BIRinBSP;
        CSSM_HRS_BIR_PTR  BIR;
    } InputBIR;
} CSSM_HRS_INPUT_BIR, *CSSM_HRS_INPUT_BIR_PTR;
```

2.1.32 CSSM_HRS_INPUT_BIR_FORM

```
typedef uint8 CSSM_HRS_INPUT_BIR_FORM;
#define CSSM_HRS_DATABASE_ID_INPUT    (1)
#define CSSM_HRS_BIR_HANDLE_INPUT     (2)
#define CSSM_HRS_FULLBIR_INPUT        (3)
```

2.1.33 CSSM_HRS_POWER_MODE

An enumeration that specifies the types of power modes the system will try to use.

```
typedef uint32 CSSM_HRS_POWER_MODE;
/* All functions available */
#define CSSM_HRS_POWER_NORMAL    (1)
/* Able to detect (for example) insertion/fingeron/person */
/* present type of events */
#define CSSM_HRS_POWER_DETECT    (2)
/* Minimum mode. all functions off */
#define CSSM_HRS_POWER_SLEEP     (3)
```

2.1.34 CSSM_HRS_QUALITY

A value indicating the quality of the biometric data in a BIR.

```
typedef sint8 CSSM_HRS_QUALITY;
```

Note: All integer values in the BIR header are little-endian.

The performance of biometrics varies with the quality of the biometric data. Since a universally accepted definition for this quality does not exist, HRS has elected to provide the following structure with the goal of framing the effect of quality on usage of the BSP (as envisioned by the BSP vendor). The scores as reported by the BSP are based on the purpose (BIR_PURPOSE) indicted by the application (for example, capture for enrollment/verify, capture for enrollment/identify, capture for verify, and so on). Additionally, the demands upon the biometric vary based on the actual customer application and/or environment (that is, a particular application usage may require higher-quality samples than would normally be required by less demanding applications).

Quality measurements are reported as an integral value in the range 0-100, except as follows:

- Value of −1: CSSM_HRS_QUALITY was not set by the BSP (reference BSP vendor's documentation for explanation).
- Value of −2: CSSM_HRS_QUALITY is not supported by the BSP.

There are two objectives in providing CSSM_HRS_QUALITY feedback to the application:

1. The primary objective is to have the BSP inform the application how suitable the biometric sample is for the purpose (CSSM_HRS_PURPOSE) specified by the application (as framed by the BSP vendor based on the use scenario intended by the BSP vendor).
2. The secondary objective is to provide the application with relative results (for example, current sample is better/worse than previous sample).

Quality scores in the range 0-100 have the following interpretation:

0-25	UNACCEPTABLE: The biometric data cannot be used for the purpose specified by the application (CSSM_HRS_PURPOSE). The biometric data must be replaced with a new sample.
26-50	MARGINAL: The biometric data will provide poor performance for the purpose specified by the application (CSSM_HRS_PURPOSE) and in most application environments will compromise the intent of the application. The biometric data should be replaced with a new sample.
51-75	ADEQUATE: The biometric data will provide good performance in most application environments based on the purpose specified by the application (CSSM_HRS_PURPOSE). The application should attempt to obtain higher quality data if the application developer anticipates demanding usage.
76-100	EXCELLENT: The biometric data will provide good performance for the purpose specified by the application (CSSM_HRS_BIR_PURPOSE). The application may want to attempt to obtain better samples if the sample quality (CSSM_HRS_QUALITY) is in the lower portion of the range (for example, 76, 77, ...) when convenient (for example, during enrollment).

2.1.35 CSSM_HRS_STREAM_CALLBACK

```
typedef CSSM_RETURN (CSSMAPI *CSSM_HRS_STREAM_CALLBACK)
    (void *StreamCallbackCtx,
    CSSM_DATA_PTR OutMessage,
    CSSM_DATA_PTR InMessage);
```

A *Callback* function that an application supplies to allow the service provider to stream data in the form of a sequence of protocol data units (messages).

Parameters

StreamCallbackCtx (input)
A generic pointer to context information that was provided by the original requester and is being returned to its originator.

OutMessage (input)
A pointer to a protocol data unit to be sent to the communication partner.

InMessage (output/optional)
A pointer to a protocol data unit to be received back from the communicating partner.

2.2 CDSA/HRS Registry Schema

The various components of the CDSA / HRS implementation are represented by records in MDS.

The following sections define the information that should be kept for each component type.

2.2.1 Data Definitions

2.2.1.1 CSSM_HRS_OPERATIONS_MASK

A mask that indicates what operations are supported by the HRS service provider.

```
typedef uint32 CSSM_HRS_OPERATIONS_MASK;
```

```
#define CSSM_HRS_CAPTURE                (0x0001)
#define CSSM_HRS_CREATETEMPLATE         (0x0002)
#define CSSM_HRS_PROCESS                (0x0004)
#define CSSM_HRS_VERIFYMATCH            (0x0008)
#define CSSM_HRS_IDENTIFYMATCH          (0x0010)
#define CSSM_HRS_ENROLL                 (0x0020)
#define CSSM_HRS_VERIFY                 (0x0040)
#define CSSM_HRS_IDENTIFY               (0x0080)
#define CSSM_HRS_IMPORT                 (0x0100)
#define CSSM_HRS_SETPOWERMODE           (0x0200)
#define CSSM_HRS_DATABASEOPERATIONS     (0x0400)
```

2.2.1.2 CSSM_HRS_OPTIONS_MASK

A mask that indicates what options are supported by the HRS Service Provider. Note that optional functions are identified within the CSSM_HRS_OPERATIONS_MASK and not repeated here.

```
typedef uint32 CSSM_HRS_OPTIONS_MASK;
```

```
#define CSSM_HRS_RAW                    (0x00000001)
    /* If set, indicates that the BSP supports the return of
       raw/audit data. */
#define CSSM_HRS_QUALITY_RAW            (0x00000002)
    /* If set, BSP supports the return of a quality value
       (in the BIR header) for raw biometric data. */
#define CSSM_HRS_QUALITY_INTERMEDIATE   (0x00000004)
    /* If set, BSP supports the return of a quality value
       (in the BIR header) for intermediate biometric data. */
#define CSSM_HRS_QUALITY_PROCESSED      (0x00000008)
    /* If set, BSP supports the return of quality value
       (in the BIR header) for processed biometric data. */
#define CSSM_HRS_APP_GUI                (0x00000010)
    /* If set, indicates that the BSP supports application
       control of the GUI. */
#define CSSM_HRS_STREAMINGDATA          (0x00000020)
    /* If set, indicates that the BSP provides GUI
       streaming data. */
#define CSSM_HRS_USERVALIDATESSAMPLES   (0x00000040)
    /* If set, user must validate each sample. */
#define CSSM_HRS_VERIFYSAMPLES          (0x00000080)
    /* If set, BSP verifies each sample. */
```

```
#define CSSM_HRS_SOURCEPRESENT         (0x00000100)
    /* If set, BSP supports detection of source presence. */
#define CSSM_HRS_PAYLOAD               (0x00001000)
    /* If set, indicates that the BSP supports payload
       carry (accepts payload during enroll/process and
       returns payload upon successful verify). */
#define CSSM_HRS_BIR_SIGN              (0x00002000)
    /* If set, BSP returns signed BIRs. */
#define CSSM_HRS_BIR_ENCRYPT           (0x00004000)
    /* If set, BSP returns encrypted BIRs. */
#define CSSM_HRS_FRR_SUPPORTED         (0x00010000)
    /* If set, indicates BSP supports the return of
       actual FRR during matching operations (Verify,
       VerifyMatch, Identify, IdentifyMatch). */
#define CSSM_HRS_ADAPTATION            (0x00020000)
    /* If set, BSP supports BIR adaptation (return
       of Verify or VerifyMatch operation). */
#define CSSM_HRS_BINNING               (0x00040000)
    /* If set, BSP supports binning (parameter used
       in Identify and IdentifyMatch operations). */
#define CSSM_HRS_DEFAULTDATABASE       (0x00080000)
    /* If set, BSP supports a default database. */
#define CSSM_HRS_LOCAL_BSP             (0x01000000)
    /* If set, BSP can  operate in standalone mode. */
#define CSSM_HRS_CLIENT_BSP            (0x02000000)
    /* If set, BSP can operate as a Client
       (that is, can Capture). */
#define CSSM_HRS_SERVER_BSP            (0x04000000)
    /* If set, BSP can operate as a Server. */
#define CSSM_HRS_STREAMINGCALLBACK     (0x08000000)
    /* If set, BSP supports streaming callbacks and
       StreamInputOutput. */
#define CSSM_HRS_PROGRESS              (0x10000000)
    /* If set, BSP supports the return of progress. */
#define CSSM_HRS_SELFCONTAINEDDEVICE   (0x20000000)
    /* If set, BSP is supporting a self-contained
       device. */
```

2.2.2 Component Schema

2.2.2.1 Primary HRS Service Provider Relation

The service provider schema describes capabilities of an HRS service provider module.

Field Name	Field Data Type	Comment
ModuleID	STRING	GUID uniquely identifying HRS SP.
SSID	UINT32	4-byte Subservice ID of attached sensor device.
ModuleName	STRING	Filename of HRS Module .
ProductVersion	STRING	HRS SP product version string (in dotted high / low format; e.g., 2.0).
Vendor	STRING	Service provider vendor name in ASCII text.
HRSSpecVersion	STRING	Highest compatible HRS Spec Version (in dotted high / low format; e.g., 2.0).
SupportedFormats	MULTIUINT32	An array of 2-byte integer pairs, each pair specifying a supported biometric data format (CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT).
FactorsMask	UINT32	A mask which indicates what forms of authentication are supported (CSSM_HRS_BIR_AUTH_FACTORS).
Operations	UINT32	Operations supported by service provider (CSSM_HRS_OPERATIONS_MASK).
Options	UINT32	Options supported by the BSP (CSSM_HRS_OPTIONS_MASK).
PayloadPolicy	UINT32	Threshold setting (minimum FAR value) used to determine when to release a payload (CSSM_HRS_FAR).
MaxPayloadSize	UINT32	Maximum size in bytes of a payload.
DefaultVerifyTimeout	SINT32	Default timeout value (in milliseconds) used by a BSP for verify operations when no timeout is set by the application.
DefaultIdentifyTimeout	SINT32	Default timeout value (in milliseconds) used by a BSP for identify operations when no timeout is set by the application.
DefaultCaptureTimeout	SINT32	Default timeout value (in milliseconds) used by a BSP for capture operations when no timeout is set by the application.
DefaultEnrollTimeout	SINT32	Default timeout value (in milliseconds) used by a BSP for enroll operations when no timeout is set by the application.
MAXBSPDBsize	UINT32	Maximum size of a BSP owned (internal) database. If NULL, no BSP database exists.
MaxIdentify	UINT32	Largest population supported by Identify function. Unlimited = FFFFFFFF

2.2.2.2 Biometric Device Relation

The information in the biometric device registry entry is updated each time a biometric device is attached to or removed from the service provider.

Field Name	Field Data Type	Comment
ModuleID	STRING	GUID (in string format) uniquely identifying service provider module.
SSID	UINT32	4-byte device ID.
SupportedFormats	MULTIUINT32	BIR Formats supported by BSP+device (CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT).
SupportedEvents	UINT32	A CSSM_HRS_MODULE_EVENT_MASK indicating which types of events are supported.
DeviceVendor	STRING	Unicode text name of device vendor.
DeviceDescription	STRING	Unicode text description of the biometric device.
DeviceSerialNumber	STRING	Serial Number of biometric device.
DeviceHardwareVersion	STRING	Device hardware version string (in dotted high / low format; e.g., 2.0).
DeviceFirmwareVersion	STRING	Device Firmware version string (in dotted high / low format; e.g., 2.0).
AuthenticatedDevice	BOOL	An indication of whether the device has been authenticated.

2.3 HRS Error Codes

2.3.1 Configurable HRS Error Code Constants

The following constant can be configured on a per-platform basis:

```
#define CSSM_HRS_BASE_ERROR \
    (CSSM_AC_BASE_ERROR + CSSM_ERRCODE_MODULE_EXTENT)
#define CSSM_HRS_PRIVATE_ERROR \
    (CSSM_HRS_BASE_ERROR + CSSM_ERROR_CUSTOM_OFFSET)
```

2.3.2 HRS Error Values Derived from Common Error Codes

```
#define CSSMERR_HRS_INTERNAL_ERROR \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_INTERNAL_ERROR)

#define CSSMERR_HRS_MEMORY_ERROR \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_MEMORY_ERROR)

#define CSSMERR_HRS_INVALID_POINTER \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_INVALID_POINTER)

#define CSSMERR_HRS_INVALID_INPUT_POINTER \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_INVALID_INPUT_POINTER)

#define CSSMERR_HRS_INVALID_OUTPUT_POINTER \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_INVALID_OUTPUT_POINTER)

#define CSSMERR_HRS_FUNCTION_NOT_IMPLEMENTED \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_FUNCTION_NOT_IMPLEMENTED)

#define CSSMERR_HRS_OS_ACCESS_DENIED \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_OS_ACCESS_DENIED)

#define CSSMERR_HRS_FUNCTION_FAILED \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_FUNCTION_FAILED)

#define CSSMERR_HRS_INVALID_DATA \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_INVALID_DATA)

#define CSSMERR_HRS_INVALID_DB_HANDLE \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_INVALID_DB_HANDLE)
```

2.3.3 HRS-Specific Error Values

```
#define CSSM_HRS_BASE_HRS_ERROR \
    (CSSM_HRS_BASE_ERROR+CSSM_ERRCODE_COMMON_EXTENT)

#define CSSMERR_HRS_UNABLE_TO_CAPTURE (CSSM_HRS_BASE_HRS_ERROR+1)
    /* The BSP is unable to capture raw samples from the device. */

#define CSSMERR_HRS_TOO_MANY_HANDLES (CSSM_HRS_BASE_HRS_ERROR+2)
    /* The BSP has no more space to allocate BIR handles. */

#define CSSMERR_HRS_TIMEOUT_EXPIRED (CSSM_HRS_BASE_HRS_ERROR+3)
    /* The Function has been terminated because the timeout
       value has expired. */

#define CSSMERR_HRS_INVALID_BIR (CSSM_HRS_BASE_HRS_ERROR+4)
    /* The input BIR is invalid for the purpose required. */
```

```
#define CSSMERR_HRS_BIR_SIGNATURE_FAILURE (CSSM_HRS_BASE_HRS_ERROR+5)
    /* The BSP could not validate the signature on the BIR. */

#define CSSMERR_HRS_UNABLE_TO_WRAP_PAYLOAD (CSSM_HRS_BASE_HRS_ERROR+6)
    /* The BSP is unable to include the payload in the new BIR. */

#define CSSMERR_HRS_NO_INPUT_BIRS (CSSM_HRS_BASE_HRS_ERROR+8)
    /* The identify population is NULL. */

#define CSSMERR_HRS_UNSUPPORTED_FORMAT (CSSM_HRS_BASE_HRS_ERROR+9)
    /* The BSP does not support the data form for the
       Import function. */

#define CSSMERR_HRS_UNABLE_TO_IMPORT (CSSM_HRS_BASE_HRS_ERROR+10)
    /* The BSP was unable to construct a BIR from the input data. */

#define CSSMERR_HRS_FUNCTION_NOT_SUPPORTED \
    (CSSM_HRS_BASE_HRS_ERROR+12)
    /* The BSP does not support this operation. */

#define CSSMERR_HRS_INCONSISTENT_PURPOSE (CSSM_HRS_BASE_HRS_ERROR+13)
    /* The purpose recorded in the BIR, and the requested
       purpose, are inconsistent with the function being
       performed. */

#define CSSMERR_HRS_BIR_NOT_FULLY_PROCESSED \
    (CSSM_HRS_BASE_HRS_ERROR+14)
    /* The function requires a fully-processed BIR. */

#define CSSMERR_HRS_PURPOSE_NOT_SUPPORTED (CSSM_HRS_BASE_HRS_ERROR+15)
    /* The BSP does not support the requested purpose. */

#define CSSMERR_HRS_INVALID_HANDLE (CSSM_HRS_BASE_HRS_ERROR+16)
    /* No BIR exists with the requested handle. */

#define CSSMERR_HRS_UNABLE_TO_OPEN_DATABASE \
    (CSSM_HRS_BASE_HRS_ERROR+256)
    /* BSP is unable to open the specified database. */

#define CSSMERR_HRS_DATABASE_IS_LOCKED (CSSM_HRS_BASE_HRS_ERROR+257)
    /* The database cannot be opened for the access
       requested because it is locked. */

#define CSSMERR_HRS_DATABASE_DOES_NOT_EXIST \
    (CSSM_HRS_BASE_HRS_ERROR+258)
    /* The specified database name does not exist. */

#define CSSMERR_HRS_DATABASE_ALREADY_EXISTS \
    (CSSM_HRS_BASE_HRS_ERROR+259)
    /* Create failed because the database already exists. */

#define CSSMERR_HRS_INVALID_DATABASE_NAME (CSSM_HRS_BASE_HRS_ERROR+260)
    /* The database name is invalid. */

#define CSSMERR_HRS_RECORD_NOT_FOUND (CSSM_HRS_BASE_HRS_ERROR+261)
    /* No record exists with the requested key. */

#define CSSMERR_HRS_CURSOR_IS_INVALID (CSSM_HRS_BASE_HRS_ERROR+262)
    /* The specified cursor is invalid. */
```

```
#define CSSMERR_HRS_DATABASE_IS_OPEN (CSSM_HRS_BASE_HRS_ERROR+263)
    /* The database is already open. */

#define CSSMERR_HRS_INVALID_ACCESS_REQUEST \
    (CSSM_HRS_BASE_HRS_ERROR+264)
    /* The access type is unrecognized. */

#define CSSMERR_HRS_END_OF_DATABASE (CSSM_HRS_BASE_HRS_ERROR+265)
    /* End of database has been reached. */

#define CSSMERR_HRS_UNABLE_TO_CREATE_DATABASE \
    (CSSM_HRS_BASE_BSP_ERROR+266)
    /* BSP cannot create database. */

#define CSSMERR_HRS_UNABLE_TO_CLOSE_DATABASE \
    (CSSM_HRS_BASE_BSP_ERROR+267)
    /* BSP cannot close database. */

#define CSSMERR_HRS_UNABLE_TO_DELETE_DATABASE \
    (CSSM_HRS_BASE_BSP_ERROR+268)
    /* BSP cannot delete database. */
```

2.4 BSP Operations

2.4.1 Handle Operations

This section gives the definitions for the *BSP Handle* operations.

NAME

CSSM_HRS_FreeBIRHandle, HRS_FreeBIRHandle

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_FreeBIRHandle
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_HANDLE  Handle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_FreeBIRHandle
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_HANDLE  Handle);
```

DESCRIPTION

Frees the memory and resources associated with the specified BIR Handle. The associated BIR is no longer referenceable through that handle. If necessary, the application must make the BIR persistent either in an HRS-managed database or an application-managed database before freeing the handle.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

Handle (input)
The BIR Handle to be freed.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_GetBIRFromHandle, HRS_GetBIRFromHandle

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_GetBIRFromHandle
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_HANDLE  Handle,
    CSSM_HRS_BIR_PTR *BIR);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_GetBIRFromHandle
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_HANDLE  Handle,
    CSSM_HRS_BIR_PTR *BIR);
```

DESCRIPTION

Retrieves the BIR associated with a BIR handle. The handle is invalidated. The HRS service provider allocates the storage for both the retrieved BIR structure and its data members, using the application's memory allocation callback function.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

Handle (input)
The handle of the BIR to be retrieved.

BIR (output)
The retrieved BIR.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_GetHeaderFromHandle, HRS_GetHeaderFromHandle

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_GetHeaderFromHandle
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_HANDLE  Handle,
    CSSM_HRS_BIR_HEADER_PTR  Header);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_GetHeaderFromHandle
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_HANDLE  Handle,
    CSSM_HRS_BIR_HEADER_PTR  Header);
```

DESCRIPTION

Retrieves the BIR header identified by *Handle*. The BIR Handle is not freed by the HRS service provider.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

Handle (input)
The handle of the BIR whose header is to be retrieved.

Header (output)
The header of the specified BIR.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

2.4.2 Callback and Event Operations

This section gives the definitions for the *BSP Callback and Event* operations.

NAME

CSSM_HRS_EnableEvents, HRS_EnableEvents

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_EnableEvents
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_MODULE_EVENT_MASK *Events);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_EnableEvents
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_MODULE_EVENT_MASK *Events);
```

DESCRIPTION

This function enables the events (indicated by the Events mask) from the attached HRS service provider in the current process. All other events from this HRS service provider are disabled for this process.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

Events (input)
A pointer to a mask indicating which events to enable.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_SetGUICallbacks, HRS_SetGUICallbacks

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_SetGUICallbacks
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_GUI_STREAMING_CALLBACK  GuiStreamingCallback,
    void *GuiStreamingCallbackCtx,
    CSSM_HRS_GUI_STATE_CALLBACK  GuiStateCallback,
    void *GuiStateCallbackCtx);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_SetGUICallbacks
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_GUI_STREAMING_CALLBACK  GuiStreamingCallback,
    void *GuiStreamingCallbackCtx,
    CSSM_HRS_GUI_STATE_CALLBACK  GuiStateCallback,
    void *GuiStateCallbackCtx);
```

DESCRIPTION

This function allows the application to establish callbacks so that the application may control the ‘‘look-and-feel’’ of the biometric user interface.

Note that not all HRS service providers provide streaming data.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

GuiStreamingCallback (input)
A pointer to an application callback to deal with the presentation of biometric streaming data.

GuiStreamingCallbackCtx (input)
A generic pointer to context information provided by the application that will be presented on the callback.

GuiStateCallback (input)
A pointer to an application callback to deal with GUI state changes.

GuiStateCallbackCtx (input)
A generic pointer to context information provided by the application that will be presented on the callback.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_CancelGUICallbacks, HRS_CancelGUICallbacks

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_CancelGUICallbacks
    (CSSM_HRS_HANDLE  ModuleHandle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_CancelGUICallbacks
    (CSSM_HRS_HANDLE  ModuleHandle);
```

DESCRIPTION

This function cancels *GUICallbacks* if they have been set. A *GUICallback* should be canceled before the service provider is detached.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_SetStreamCallback, HRS_SetStreamCallback

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_SetStreamCallback
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_STREAM_CALLBACK  StreamCallback,
    void *StreamCallbackCtx);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_SetStreamCallback
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_STREAM_CALLBACK  StreamCallback,
    void *StreamCallbackCtx);
```

DESCRIPTION

This function allows the application to establish a callback for client/server communication. The callback allows the HRS service provider to send a protocol message to its partner service provider, and to receive a protocol message in exchange.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

StreamCallback (input)
A pointer to an application callback to deal with the client/server transmission of protocol data units between HRS service providers.

StreamCallbackCtx (input)
A generic pointer to context information provided by the application that will be presented on the callback.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_CancelStreamCallbacks, HRS_CancelStreamCallbacks

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_CancelStreamCallbacks
    (CSSM_HRS_HANDLE  ModuleHandle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_CancelStreamCallbacks
    (CSSM_HRS_HANDLE  ModuleHandle);
```

DESCRIPTION

This function cancels the *StreamCallback* if it has been set. A *StreamCallback* should be canceled before the service provider is detached.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_StreamInputOutput, HRS_StreamInputOutput

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_StreamInputOutput
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_DATA_PTR  InMessage,
    CSSM_DATA_PTR  OutMessage);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_StreamInputOutput
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_DATA_PTR  InMessage,
    CSSM_DATA_PTR  OutMessage);
```

DESCRIPTION

This function allows the application to pass a protocol data unit into the HRS service provider from the partner HRS service provider, and to obtain a response message to return to the partner; see Section 1.4 (on page 4).

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

InMessage (input)
InMessage contains a protocol data unit from the partner HRS service provider.

OutMessage (output)
OutMessage contains a protocol data unit to be sent back to the partner HRS service provider. If the parameter is NULL, there is no message to return.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

2.4.3 Biometric Operations

This section gives the definitions for the *BSP Biometric* operations.

NAME

CSSM_HRS_Capture, HRS_Capture

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Capture
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_PURPOSE  Purpose,
    CSSM_HRS_BIR_HANDLE_PTR  CapturedBIR,
    sint32  Timeout,
    CSSM_HRS_BIR_HANDLE_PTR  AuditData);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Capture
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_PURPOSE  Purpose,
    CSSM_HRS_BIR_HANDLE_PTR  CapturedBIR,
    sint32  Timeout,
    CSSM_HRS_BIR_HANDLE_PTR  AuditData);
```

DESCRIPTION

This function captures samples for the purpose specified, and returns either an ‘‘intermediate’’ type BIR (if the *Process* function needs to be called), or a ‘‘processed’’ BIR (if not). The *Purpose* is recorded in the header of the *CapturedBIR*. If *AuditData* is non-NULL, a BIR of type ‘‘raw’’ may be returned. The function returns handles to whatever data is collected, and all local operations can be completed through use of the handles.

If the application needs to acquire the data either to store it in a database or to send it to a server, the application can retrieve the data with the *HRS_GetBIRFromHandle*( ) function.

The application may request control of the GUI ‘‘look-and-feel’’ by providing a GUI callback pointer in *HRS_SetGUICallbacks*( ).

Capture serializes use of the device. If two or more applications are racing for the device, the losers will wait until the timeout expires. This serialization takes place in all functions that capture data.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

Purpose (input)
A value indicating the purpose of the biometric data capture.

CapturedBIR (output)
A handle to a BIR containing captured data. This data is either an ‘‘intermediate’’ type BIR (which can only be used by either the *Process* or *CreateTemplate* functions, depending on the purpose), or a ‘‘processed’’ BIR (which can be used directly by *VerifyMatch* or *IdentifyMatch*, depending on the purpose).

Timeout (input)
An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive

number. A −1 value means the service provider's default timeout value will be used.

AuditData (output/optional)
A handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected. Not all BSPs support the collection of audit data. A service provider may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE indicating not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE indicating no audit data is available.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_TIMEOUT_EXPIRED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_CAPTURE

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_CreateTemplate, HRS_CreateTemplate

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_CreateTemplate
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_INPUT_BIR *CapturedBIR,
    const CSSM_HRS_INPUT_BIR *StoredTemplate,
    CSSM_HRS_BIR_HANDLE_PTR  NewTemplate,
    const CSSM_DATA *Payload);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_CreateTemplate
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_INPUT_BIR *CapturedBIR,
    const CSSM_HRS_INPUT_BIR *StoredTemplate,
    CSSM_HRS_BIR_HANDLE_PTR  NewTemplate,
    const CSSM_DATA *Payload);
```

DESCRIPTION

This function takes a BIR containing raw biometric data for the purpose of creating a new enrollment template. A new BIR is constructed from the *CapturedBIR*, and (optionally) it may perform an adaptation based on an existing *StoredTemplate*. The old *StoredTemplate* remains unchanged. If the *StoredTemplate* contains a payload, the payload is not copied into the *NewTemplate*. If the *NewTemplate* needs a payload, then that *Payload* must be presented as an argument to the function.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

CapturedBIR (input)
The captured BIR or its handle.

StoredTemplate (input/optional)
Optionally, the template to be adapted, or its key in a database, or its handle.

NewTemplate (output)
A handle to a newly created template that is derived from the *CapturedBIR* and (optionally) the *StoredTemplate*.

Payload (input/optional)
A pointer to data that will be wrapped inside the newly created template. This parameter is ignored, if NULL.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_INVALID_BIR
CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_WRAP_PAYLOAD

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_Process, HRS_Process

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Process
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_INPUT_BIR *CapturedBIR,
    CSSM_HRS_BIR_HANDLE_PTR  ProcessedBIR);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Process
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_INPUT_BIR *CapturedBIR,
    CSSM_HRS_BIR_HANDLE_PTR  ProcessedBIR);
```

DESCRIPTION

This function processes the intermediate data captured via a call to *HRS_Capture*( ) for the purpose of either verification or identification. If the processing capability is in the attached service provider, a ‘‘processed’’ BIR is returned; otherwise, *ProcessedBIR* is set to NULL.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

CapturedBIR (input)
The captured BIR or its handle.

ProcessedBIR (output)
A handle for the newly constructed ‘‘processed’’ BIR, NULL.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_INVALID_BIR
CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_WRAP_PAYLOAD

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_VerifyMatch, HRS_VerifyMatch

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_VerifyMatch
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_FAR *MaxFARRequested,
    const CSSM_HRS_FRR *MaxFRRRequested,
    const CSSM_BOOL *FARPrecedence,
    const CSSM_HRS_INPUT_BIR *ProcessedBIR,
    const CSSM_HRS_INPUT_BIR *StoredTemplate,
    CSSM_HRS_BIR_HANDLE *AdaptedBIR,
    CSSM_BOOL *Result,
    CSSM_HRS_FAR_PTR  FARAchieved,
    CSSM_HRS_FRR_PTR  FRRAchieved,
    CSSM_DATA_PTR *Payload);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_VerifyMatch
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_FAR *MaxFARRequested,
    const CSSM_HRS_FRR *MaxFRRRequested,
    const CSSM_BOOL *FARPrecedence,
    const CSSM_HRS_INPUT_BIR *ProcessedBIR,
    const CSSM_HRS_INPUT_BIR *StoredTemplate,
    CSSM_HRS_BIR_HANDLE *AdaptedBIR,
    CSSM_BOOL *Result,
    CSSM_HRS_FAR_PTR  FARAchieved,
    CSSM_HRS_FRR_PTR  FRRAchieved,
    CSSM_DATA_PTR *Payload);
```

DESCRIPTION

This function performs a verification (1-to-1) match between two BIRs—the *ProcessedBIR* and the *StoredTemplate*. The *ProcessedBIR* is the ‘‘processed’’ BIR constructed specifically for this verification. The *StoredTemplate* was created at enrollment. The application must request a maximum FAR value for a successful match, and may also (optionally) request a maximum FRR for a successful match. If a maximum FRR value is provided, the application must also indicate (via the *FARPrecedence* parameter) which one takes precedence. The Boolean *Result* indicates whether verification was successful or not, and the *FARAchieved* is a FAR value indicating how closely the BIRs actually matched.

The service provider may optionally return the corresponding FRR that was achieved, through the *FRRAchieved* return parameter.

By setting the *AdaptedBIR* pointer to non-NULL, the application can request that a BIR be constructed by adapting the *StoredTemplate* using the *ProcessedBIR*. A new handle is returned to the *AdaptedBIR*. If the *StoredTemplate* contains a *Payload*, the *Payload* may be returned upon successful verification if the *FARAchieved* is sufficiently stringent. This is controlled by the policy of the service provider.

If the match is successful, an attempt may be made to adapt the *StoredTemplate* with information taken from the *ProcessedBIR*. (Not all service providers perform adaptation.) The resulting *AdaptedBIR* should now be considered an optimal enrollment template, and be saved in the enrollment database. It is up to the application whether or not it uses or discards this data. It is important to note that adaptation may not occur in all cases.

In the event of an adaptation, this function stores the handle to the new BIR in the memory pointed to by the *AdaptedBIR* parameter.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

MaxFARRequested (input)
The requested FAR criterion for successful verification.

MaxFRRRequested (input/optional)
The requested FRR criterion for successful verification. A NULL pointer indicates that this criterion is not provided.

FARPrecedence (input)
If both criteria are provided, this parameter indicates which takes precedence: CSSM_TRUE for FAR, CSSM_FALSE for FRR.

ProcessedBIR (input)
The BIR to be verified, or its handle.

StoredTemplate (input)
The BIR to be verified against, or its key in a database, or its handle.

AdaptedBIR (output/optional)
A pointer to the handle of the adapted BIR. This parameter can be NULL if an Adapted BIR is not desired. Not all service providers support the adaptation of BIRs. The function may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that adaptation was not possible.

Result (output)
A pointer to a Boolean value indicating (CSSM_TRUE/CSSM_FALSE) whether the BIRs matched or not according to the specified criteria.

FARAchieved (output)
A pointer to an FAR value indicating the closeness of the match.

FRRAchieved (output/optional)
A pointer to an FRR value indicating the closeness of the match.

Payload (output/optional)
If the *StoredTemplate* contains a payload, it is returned in an allocated CSSM_DATA structure if the *FARAchieved* satisfies the policy of the service provider.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_ADAPTATION_NOT_SUPPORTED
CSSMERR_HRS_BIR_NOT_FULLY_PROCESSED
CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_INCONSISTENT_PURPOSE

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_IdentifyMatch, HRS_IdentifyMatch

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_IdentifyMatch
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_FAR *MaxFARRequested,
    const CSSM_HRS_FRR *MaxFRRRequested,
    const CSSM_BOOL *FARPrecedence,
    const CSSM_HRS_INPUT_BIR *ProcessedBIR,
    const CSSM_HRS_IDENTIFY_POPULATION *Population,
    CSSM_BOOL  Binning,
    uint32  MaxNumberOfResults,
    uint32 *NumberOfResults,
    CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
    sint32  Timeout);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_IdentifyMatch
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_FAR *MaxFARRequested,
    const CSSM_HRS_FRR *MaxFRRRequested,
    const CSSM_BOOL *FARPrecedence,
    const CSSM_HRS_INPUT_BIR *ProcessedBIR,
    const CSSM_HRS_IDENTIFY_POPULATION *Population,
    CSSM_BOOL  Binning,
    uint32  MaxNumberOfResults,
    uint32 *NumberOfResults,
    CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
    sint32  Timeout);
```

DESCRIPTION

This function performs an identification (1-to-many) match between a *ProcessedBIR* and a set of stored BIRs. The *ProcessedBIR* is the ‘‘processed’’ BIR captured specifically for this identification. The population that the match takes place against can be presented in one of three ways:

1. In a database identified by an open database handle
2. Input in an array of BIRs
3. In the ‘‘default’’ database of the BSP (possibly stored in the biometric device)

The application must request a maximum FAR value criterion for a successful match, and may also (optionally) request a maximum FRR criterion for a successful match. If a maximum FRR value is provided, the application must also indicate, via the *FARPrecedence* parameter, which criteria takes precedence. The *FARAchieved* and, optionally, the *FRRAchieved* are returned for each result in the *Candidate* array.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
: The handle of the attached HRS service provider.

MaxFARRequested (input)
The requested FAR criterion for successful verification.

MaxFRRRequested (input / optional)
The requested FRR criterion for successful verification. A NULL pointer indicates that this criterion is not provided.

FARPrecedence (input)
If both criteria are provided, this parameter indicates which takes precedence: CSSM_TRUE for FAR, CSSM_FALSE for FRR.

ProcessedBIR (input)
The BIR to be identified.

Population (input)
The population of BIRs against which the *Identify* match is performed.

Binning (input)
A Boolean indicating whether *Binning* is on or off. Binning is a search-optimization technique that the service provider may employ. It is based on searching the population according to the intrinsic characteristics of the biometric data. While it may improve the speed of the *Match* operation, it may also increase the probability of missing a candidate.

MaxNumberOfResults (input)
Specifies the maximum number of match candidates to be returned as a result of the 1:N match. A value of zero is a request for all candidates.

NumberOfResults (output)
Specifies the number of candidates returned in the *Candidates* array as a result of the 1:N match.

Candidates (output)
A pointer to an array of CSSM_HRS_CANDIDATE_PTRs corresponding to the BIRs identified as a result of the match process (that is, indices associated with BIRs found to exceed the match threshold). This list is in rank order, with the highest scoring record being first. If no matches are found, this pointer will be set to NULL. If the *Population* was presented in a database, the IDs are database IDs; if the set was presented in an in-memory array, the IDs are indexes into the array.

Timeout (input)
An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no candidate list. This value can be any positive number. A –1 value means the service provider's default timeout value will be used.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_BIR_NOT_FULLY_PROCESSED
CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_FUNCTION_NOT_SUPPORTED
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_NO_INPUT_BIRS
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TIMEOUT_EXPIRED

See HRS Error Codes in Section 2.3 (on page 25).

NOTES

Not all service providers support 1:N identification.

Depending on the service provider and the location and size of the database to be searched, this operation can take a significant amount of time to perform

The number of match candidates found by the service provider is dependent on the actual FAR used.

NAME

CSSM_HRS_Enroll, HRS_Enroll

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Enroll
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_PURPOSE  Purpose,
    const CSSM_HRS_INPUT_BIR *StoredTemplate,
    CSSM_HRS_BIR_HANDLE_PTR  NewTemplate,
    const CSSM_DATA *Payload,
    sint32  Timeout
    CSSM_HRS_BIR_HANDLE_PTR  AuditData);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Enroll
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_BIR_PURPOSE  Purpose,
    const CSSM_HRS_INPUT_BIR *StoredTemplate,
    CSSM_HRS_BIR_HANDLE_PTR  NewTemplate,
    const CSSM_DATA *Payload,
    sint32  Timeout
    CSSM_HRS_BIR_HANDLE_PTR  AuditData);
```

DESCRIPTION

This function captures biometric data from the attached device for the purpose of creating a *ProcessedBIR* for the purpose of enrollment. The *Enroll* function may be split between client and server if a streaming callback has been set. Either the client or the server can initiate the operation.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

Purpose (input)
A value indicating the desired purpose of *Enrollment*.

StoredTemplate (input/optional)
Optionally, the BIR to be adapted, or its key in a database, or its handle.

NewTemplate (output)
A handle to a newly created template that is derived from the new raw samples and (optionally) the *StoredTemplate*.

Payload (input/optional)
A pointer to data that will be wrapped inside the newly created template. This parameter is ignored, if NULL.

Timeout (input)
An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A −1 value means the BSP's default timeout value will be used.

AuditData (output/optional)

A handle to a BIR containing biometric audit data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected. Not all HRS service providers support the collection of audit data. An HRS service provider may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate *AuditData* is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that no audit data is available.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TIMEOUT_EXPIRED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_CAPTURE
CSSMERR_HRS_UNABLE_TO_WRAP_PAYLOAD

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_Verify, HRS_Verify

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Verify
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_FAR *MaxFARRequested,
    const CSSM_HRS_FRR *MaxFRRRequested,
    const CSSM_BOOL *FARPrecedence,
    const CSSM_HRS_INPUT_BIR *StoredTemplate,
    CSSM_HRS_BIR_HANDLE_PTR  AdaptedBIR,
    CSSM_BOOL *Result,
    CSSM_HRS_FAR_PTR  FARAchieved,
    CSSM_HRS_FRR_PTR  FRRAchieved,
    CSSM_DATA_PTR *Payload,
    sint32  Timeout,
    CSSM_HRS_BIR_HANDLE_PTR  AuditData);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Verify
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_FAR *MaxFARRequested,
    const CSSM_HRS_FRR *MaxFRRRequested,
    const CSSM_BOOL *FARPrecedence,
    const CSSM_HRS_INPUT_BIR *StoredTemplate,
    CSSM_HRS_BIR_HANDLE_PTR  AdaptedBIR,
    CSSM_BOOL *Result,
    CSSM_HRS_FAR_PTR  FARAchieved,
    CSSM_HRS_FRR_PTR  FRRAchieved,
    CSSM_DATA_PTR *Payload,
    sint32  Timeout,
    CSSM_HRS_BIR_HANDLE_PTR  AuditData);
```

DESCRIPTION

This function captures biometric data from the attached device, and compares it against the *StoredTemplate*. The application must request a maximum FAR value criterion for a successful match, and may also (optionally) request a maximum FRR criterion for a successful match. If a maximum FRR value is provided, the application must also indicate via the *FARPrecedence* parameter, which criteria takes precedence. The Boolean *Result* indicates whether verification was successful or not, and the *FARAchieved* is a FAR value indicating how closely the BIRs actually matched.

The service provider may optionally return the corresponding FRR that was achieved through the *FRRAchieved* return parameter.

If the *StoredTemplate* contains a payload, the *Payload* may be returned upon successful verification. Optionally, a new *AdaptedBIR* may be constructed.

The *Verify* function may be split between client and server if a streaming callback has been set. Either the client or the server can initiate the operation.

If the match is successful, an attempt may be made to adapt the *StoredTemplate* with information taken from the *ProcessedBIR*. (Not all service providers perform adaptation.) The resulting *AdaptedBIR* should now be considered an optimal enrollment, and be saved in the enrollment database. It is up to the application whether or not it uses or discards this data. It is important to note that adaptation may not occur in all cases.

In the event of an adaptation, this function stores the handle to the new BIR in the memory pointed to by the *AdaptedBIR* parameter.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

MaxFARRequested (input)
The requested FAR criterion for successful verification.

MaxFRRRequested (input/optional)
The requested FRR criterion for successful verification. A NULL pointer indicates that this criterion is not provided.

FARPrecedence (input)
If both criteria are provided, this parameter indicates which takes precedence: CSSM_TRUE for FAR, CSSM_FALSE for FRR.

StoredTemplate (input)
The BIR to be verified against, or its key in a database, or its handle.

AdaptedBIR (output/optional)
A pointer to the handle of the adapted BIR. This parameter can be NULL if an adapted BIR is not desired. Not all HRS service providers support the adaptation of BIRs. The function may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that adaptation was not possible.

Result (output)
A pointer to a Boolean value indicating (CSSM_TRUE/CSSM_FALSE) whether the BIRs matched or not, according to the specified criteria.

FARAchieved (output)
A pointer to an FAR value indicating the closeness of the match.

FRRAchieved (output/optional)
A pointer to an FRR value indicating the closeness of the match.

Payload (output/optional)
If the *StoredTemplate* contains a payload, it is returned in an allocated CSSM_DATA structure if the *FARAchieved* satisfies the policy of the service provider.

Timeout (input)
An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A −1 value means the service provider's default timeout value will be used.

AuditData (output/optional)
A handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected. Not all service providers support the collection of audit data. An HRS service

provider may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate *AuditData* is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that no audit data is available.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TIMEOUT_EXPIRED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_CAPTURE

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_Identify, HRS_Identify

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Identify
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_FAR *MaxFARRequested,
    const CSSM_HRS_FRR *MaxFRRRequested,
    const CSSM_BOOL *FARPrecedence,
    const CSSM_HRS_IDENTIFY_POPULATION *Population,
    CSSM_BOOL  Binning,
    uint32  MaxNumberOfResults,
    uint32 *NumberOfResults,
    CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
    sint32  Timeout,
    CSSM_HRS_BIR_HANDLE_PTR  AuditData);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Identify
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_FAR *MaxFARRequested,
    const CSSM_HRS_FRR *MaxFRRRequested,
    const CSSM_BOOL *FARPrecedence,
    const CSSM_HRS_IDENTIFY_POPULATION *Population,
    CSSM_BOOL  Binning,
    uint32  MaxNumberOfResults,
    uint32 *NumberOfResults,
    CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
    sint32  Timeout,
    CSSM_HRS_BIR_HANDLE_PTR  AuditData);
```

DESCRIPTION

This function captures biometric data from the attached device, and compares it against the *Population*. The application must request a maximum FAR value criterion for a successful match, and may also (optionally) request a maximum FRR criterion for a successful match. If a maximum FRR value is provided, the application must also indicate via the *FARPrecedence* parameter, which criteria takes precedence.

The function returns a number of candidates from the population that match according to the specified criteria, and the *FARAchieved* and, optionally, the *FRRAchieved* are returned for each result in the *Candidate* array.

The *Identify* function may be split between client and server if a streaming callback has been set. Either the client or the server can initiate the operation.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

MaxFARRequested (input)
The requested FAR criterion for successful identification.

MaxFRRRequested (input/optional)
The requested FRR criterion for successful identification. A NULL pointer indicates that this criterion is not provided.

FARPrecedence (input)
If both criteria are provided, this parameter indicates which takes precedence: CSSM_TRUE for FAR, CSSM_FALSE for FRR.

Population (input)
The population of *Templates* against which the *Identify* match is performed.

Binning (input)
A boolean value indicating whether *Binning* is on or off. *Binning* is a search optimization technique that the BSP may employ. It is based on searching the population according to the intrinsic characteristics of the biometric data. While it may improve the speed of the *Match* operation, it may also increase the probability of missing a candidate.

MaxNumberOfResults (input)
Specifies the maximum number of match candidates to be returned as a result of the 1:N match. A value of zero is a request for all candidates.

NumberOfResults (output)
Specifies the number of candidates returned in the *Candidates* array as a result of the 1:N match.

Candidates (output)
A pointer to an array of CSSM_HRS_CANDIDATE_PTRs corresponding to the BIRs identified as a result of the match process (that is, indices associated with BIRs found to exceed the match threshold). This list is in rank order, with the highest scoring record being first. If no matches are found, this pointer will be set to NULL. If the *Population* was presented in a database, the IDs are database IDs; if the set was presented in an in-memory array, the IDs are indexes into the array.

Timeout (input)
An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A –1 value means the service provider's default timeout value will be used.

AuditData (output/optional)
A handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected. Not all HRS service providers support the collection of audit data. A BSP may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate *AuditData* is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that no audit data is available.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_FUNCTION_NOT_SUPPORTED
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_NO_INPUT_BIRS
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TIMEOUT_EXPIRED

CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_CAPTURE

See HRS Error Codes in Section 2.3 (on page 25).

NOTES

Not all service providers support 1:N identification. See your service provider's programming manual for more details.

Depending on the service provider and the location and size of the database to be searched, this operation can take a significant amount of time to perform. Check your service provider's manual for recommended *Timeout* values.

The number of match candidates found by the service provider is dependent on the actual FAR used.

NAME

CSSM_HRS_Import, HRS_Import

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Import
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_DATA *InputData,
    CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT  InputFormat,
    CSSM_HRS_BIR_PURPOSE  Purpose,
    CSSM_HRS_BIR_HANDLE_PTR  ConstructedBIR);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Import
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_DATA *InputData,
    CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT  InputFormat,
    CSSM_HRS_BIR_PURPOSE  Purpose,
    CSSM_HRS_BIR_HANDLE_PTR  ConstructedBIR);
```

DESCRIPTION

This function imports non-realtime raw biometric data to construct a BIR for the purpose specified. *InputData* identifies the memory buffer containing the raw biometric data, while *InputFormat* identifies the form of the raw biometric data.

The function returns a handle to the *ConstructedBIR*.

If the application needs to acquire the BIR either to store it in a database or to send it to a server, the application can retrieve the data with the *HRS_GetBIRFromHandle*( ) function, or store it directly using *HRS_DbStoreBIR*( ).

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

InputData (input)
A pointer to image/stream data to import into a *ProcessedBIR*. The image/stream conforms to the standard identified by *InputFormat*.

InputFormat (input)
The format of the *InputData*.

Purpose (input)
A value indicating the *Enroll* purpose.

ConstructedBIR (output)
A handle to a BIR constructed from the imported biometric data. This BIR may be either an *Intermediate* or *Processed* BIR (as indicated in the header).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_FUNCTION_NOT_SUPPORTED
CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_IMPORT
CSSMERR_HRS_UNSUPPORTED_FORMAT

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_SetPowerMode, HRS_SetPowerMode

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_SetPowerMode
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_POWER_MODE  PowerMode);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_SetPowerMode
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_POWER_MODE  PowerMode);
```

DESCRIPTION

This function sets the device to the requested power mode if the device supports it.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

PowerMode (input)
A 32-bit value indicting the power mode to set the device to.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_FUNCTION_NOT_SUPPORTED

See HRS Error Codes in Section 2.3 (on page 25).

2.4.4 Database Operations

This section gives the definitions for the *BSP Database* operations.

NAME

CSSM_HRS_DbOpen, HRS_DbOpen

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbOpen
    (CSSM_HRS_HANDLE  ModuleHandle,
    const uint8 *DbName,
    CSSM_HRS_DB_ACCESS_TYPE  AccessRequest,
    CSSM_HRS_DB_HANDLE_PTR  DbHandle,
    CSSM_HRS_DB_CURSOR_PTR  Cursor);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbOpen
    (CSSM_HRS_HANDLE  ModuleHandle,
    const uint8 *DbName,
    CSSM_HRS_DB_ACCESS_TYPE  AccessRequest,
    CSSM_HRS_DB_HANDLE_PTR  DbHandle,
    CSSM_HRS_DB_CURSOR_PTR  Cursor);
```

DESCRIPTION

This function opens the data store with the specified name under the specified access mode. A database *Cursor* is set to point to the first record in the database.

Note that the default database (if any) is always open.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

DbName (input)
A pointer to the null-terminated string containing the name of the database.

AccessRequest (input)
An indicator of the requested access mode for the database, such as read or write.

DbHandle (output)
The handle to the opened data store. The value will be set to CSSM_HRS_DB_INVALID_HANDLE if the function fails.

Cursor (output)
A handle that can be used to iterate through the database.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_DATABASE_DOES_NOT_EXIST
CSSMERR_HRS_DATABASE_IS_LOCKED
CSSMERR_HRS_INVALID_ACCESS_REQUEST
CSSMERR_HRS_INVALID_DATABASE_NAME
CSSMERR_HRS_UNABLE_TO_OPEN_DATABASE

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbClose, HRS_DbClose

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbClose
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbClose
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle);
```

DESCRIPTION

This function closes an open database. All cursors currently set to the database are freed. The default database cannot be closed.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

DbHandle (input)
The DB handle for an open database managed by the service provider. This specifies the open database to be closed.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbCreate, HRS_DbCreate

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbCreate
    (CSSM_HRS_HANDLE  ModuleHandle,
    const uint8 *DbName,
    CSSM_HRS_DB_ACCESS_TYPE  AccessRequest,
    CSSM_HRS_DB_HANDLE_PTR  DbHandle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbCreate
    (CSSM_HRS_HANDLE  ModuleHandle,
    const uint8 *DbName,
    CSSM_HRS_DB_ACCESS_TYPE  AccessRequest,
    CSSM_HRS_DB_HANDLE_PTR  DbHandle);
```

DESCRIPTION

This function creates and opens a new database. The name of the new database is specified by the input parameter *DbName*. The newly created database is opened under the specified access mode.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

DbName (input)
A pointer to the null-terminated string containing the name of the new database.

AccessRequest (input)
An indicator of the requested access mode for the database, such as read or write.

DbHandle (output)
The handle to the newly created and open data store. The value will be set to CSSM_HRS_DB_INVALID_HANDLE if the function fails.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_DATABASE_ALREADY_EXISTS
CSSMERR_HRS_INVALID_ACCESS_REQUEST
CSSMERR_HRS_INVALID_DATABASE_NAME

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbDelete, HRS_DbDelete

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbDelete
    (CSSM_HRS_HANDLE  ModuleHandle,
    const uint8 *DbName);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbDelete
    (CSSM_HRS_HANDLE  ModuleHandle,
    const uint8 *DbName);
```

DESCRIPTION

This function deletes all records from the specified database and removes all state information associated with that database.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

DbName (input)
A pointer to the null-terminated string containing the name of the database to be deleted.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_DATABASE_IS_OPEN
CSSMERR_HRS_INVALID_DATABASE_NAME

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbSetCursor, HRS_DbSetCursor

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbSetCursor
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle,
    const CSSM_GUID *KeyValue,
    CSSM_HRS_DB_CURSOR_PTR  Cursor);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbSetCursor
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle,
    const CSSM_GUID *KeyValue,
    CSSM_HRS_DB_CURSOR_PTR  Cursor);
```

DESCRIPTION

The *Cursor* is set to point to the record indicated by the *KeyValue* in the database identified by the *DbHandle*. A NULL value will set the cursor to the first record in the database.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
: The handle of the attached HRS service provider.

DbHandle (input)
: A handle to the open database.

KeyValue (input)
: The key into the database of the BIR to which the *Cursor* is to be set.

Cursor (output)
: A handle that can be used to iterate through the database from the retrieved record.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_RECORD_NOT_FOUND

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbFreeCursor, HRS_DbFreeCursor

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbFreeCursor
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_CURSOR_PTR  Cursor);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbFreeCursor
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_CURSOR_PTR  Cursor);
```

DESCRIPTION

Frees memory and resources associated with the specified *Cursor*.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

Cursor (input)
The database *Cursor* to be freed.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_CURSOR_IS_INVALID

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbStoreBIR, HRS_DbStoreBIR

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbStoreBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_INPUT_BIR *BIRToStore,
    CSSM_HRS_DB_HANDLE  DbHandle,
    CSSM_GUID_PTR  Guid);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbStoreBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    const CSSM_HRS_INPUT_BIR *BIRToStore,
    CSSM_HRS_DB_HANDLE  DbHandle,
    CSSM_GUID_PTR  Guid);
```

DESCRIPTION

The BIR identified by the *BIRToStore* parameter is stored in the open database identified by the *DbHandle* parameter. If the *BIRToStore* is identified by a BIR Handle, the input BIR Handle is freed. If the *BIRToStore* is identified by a database key value, the BIR is copied to the open database.

A new GUID is assigned to the new BIR in the database, and this GUID can be used as a key value to access the BIR later.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

BIRToStore (input)
The BIR to be stored in the open database (either the BIR, or its handle, or the index to it in another open database).

DbHandle (input)
The handle to the open database.

Guid (output)
A GUID that uniquely identifies the new BIR in the database. This GUID cannot be changed. To associate a different BIR with the user, it is necessary to delete the old one, store a new one in the database, and then replace the old GUID with the new one in the application account database.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbGetBIR, HRS_DbGetBIR

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbGetBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle,
    const CSSM_GUID *KeyValue,
    CSSM_HRS_BIR_HANDLE_PTR  RetrievedBIR,
    CSSM_HRS_DB_CURSOR_PTR  Cursor);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbGetBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle,
    const CSSM_GUID *KeyValue,
    CSSM_HRS_BIR_HANDLE_PTR  RetrievedBIR,
    CSSM_HRS_DB_CURSOR_PTR  Cursor);
```

DESCRIPTION

The BIR identified by the *KeyValue* parameter in the open database identified by the *DbHandle* parameter is retrieved.

The BIR is copied into the service provider's storage and a handle to it is returned. The *Cursor* is set to point to the next record, or the first record in the database if the retrieved BIR is the last.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

DbHandle (input)
The handle to the open database.

KeyValue (input)
The key into the database of the BIR to retrieve.

RetrievedBIR (output)
A handle to the retrieved BIR.

Cursor (output)
A handle that can be used to iterate through the database from the retrieved record.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_RECORD_NOT_FOUND

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbGetNextBIR, HRS_DbGetNextBIR

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbGetNextBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_CURSOR_PTR  Cursor,
    CSSM_HRS_BIR_HANDLE_PTR  RetrievedBIR,
    CSSM_GUID_PTR  Guid);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbGetNextBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_CURSOR_PTR  Cursor,
    CSSM_HRS_BIR_HANDLE_PTR  RetrievedBIR,
    CSSM_GUID_PTR  Guid);
```

DESCRIPTION

The BIR identified by the *Cursor* parameter is retrieved. The BIR is copied into the service provider's storage, a handle to it is returned, and a pointer to the GUID that uniquely identifies the BIR in the database is returned. The *Cursor* is updated to the next record in the database, or to the first when the end of the database is reached.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

Cursor (input/output)
A handle indicating which record to retrieve.

RetrievedBIR (output)
A handle to the retrieved BIR.

Guid (output)
The GUID that uniquely identifies the retrieved BIR in the database.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_CURSOR_IS_INVALID
CSSMERR_HRS_END_OF_DATABASE

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbQueryBIR, HRS_DbQueryBIR

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbQueryBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle,
    const CSSM_HRS_INPUT_BIR *BIRToQuery,
    CSSM_GUID_PTR  Guid);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbQueryBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle,
    const CSSM_HRS_INPUT_BIR *BIRToQuery,
    CSSM_GUID_PTR  Guid);
```

DESCRIPTION

If the BIR identified by the *BIRToQuery* parameter is in the open database identified by the *DbHandle* parameter, a pointer to its GUID is returned. Otherwise, CSSMERR_HRS_RECORD_NOT_FOUND is returned.

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

DbHandle (input)
The handle to the open database.

BIRToQuery (input)
The BIR to be queried in the open database (either the BIR, or its handle, or the key to it in another open database).

Guid (output)
The GUID that uniquely identifies the BIR in the database.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_RECORD_NOT_FOUND

See HRS Error Codes in Section 2.3 (on page 25).

NAME

CSSM_HRS_DbDeleteBIR, HRS_DbDeleteBIR

SYNOPSIS

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbDeleteBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle,
    const CSSM_GUID *KeyValue);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbDeleteBIR
    (CSSM_HRS_HANDLE  ModuleHandle,
    CSSM_HRS_DB_HANDLE  DbHandle,
    const CSSM_GUID *KeyValue);
```

DESCRIPTION

The BIR identified by the *KeyValue* parameter in the open database identified by the *DbHandle* parameter is deleted from the database.

If there is a cursor set to the deleted BIR, then the cursor is moved to the next sequential BIR (or set to the start of the database if there are no more records).

PARAMETERS

The parameter definitions are the same for the API and the SPI.

ModuleHandle (input)
The handle of the attached HRS service provider.

DbHandle (input)
The handle to the open database.

KeyValue (input)
The GUID of the BIR to be deleted.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

CSSMERR_HRS_END_OF_DATABASE
CSSMERR_HRS_RECORD_NOT_FOUND

See HRS Error Codes in Section 2.3 (on page 25).

Chapter 3

HRS Service Provider Interface

3.1 HRS Function Pointer Table

This structure defines the function table for all the HRS functions that a service provider can return to the CSSM Framework on *CSSM_ModuleAttach*( ).

The CSSM Framework uses these pointers to dispatch corresponding application programming interface functions to the HRS Service Provider for processing.

```
#define CSSMHRI CSSMAPI
typedef struct cssm_spi_hrs_funcs {
    CSSM_RETURN (CSSMAPI *FreeBIRHandle)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_BIR_HANDLE  Handle);
    CSSM_RETURN (CSSMAPI *GetBIRFromHandle)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_BIR_HANDLE  Handle,
        CSSM_HRS_BIR_PTR *BIR);
    CSSM_RETURN (CSSMAPI *GetHeaderFromHandle)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_BIR_HANDLE  Handle,
        CSSM_HRS_BIR_HEADER_PTR  Header);
    CSSM_RETURN (CSSMAPI *EnableEvents)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_MODULE_EVENT_MASK *Events);
    CSSM_RETURN (CSSMAPI *SetGUICallbacks)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_GUI_STREAMING_CALLBACK  GuiStreamingCallback,
        void *GuiStreamingCallbackCtx,
        CSSM_HRS_GUI_STATE_CALLBACK  GuiStateCallback,
        void *GuiStateCallbackCtx);
    CSSM_RETURN (CSSMAPI *CancelGUICallbacks)
        (CSSM_HRS_HANDLE  ModuleHandle);
    CSSM_RETURN (CSSMAPI *SetStreamCallback)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_STREAM_CALLBACK  StreamCallback,
        void *StreamCallbackCtx);
    CSSM_RETURN (CSSMAPI *CancelStreamCallbacks)
        (CSSM_HRS_HANDLE  ModuleHandle);
    CSSM_RETURN (CSSMAPI *StreamInputOutput)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_DATA_PTR  InMessage,
        CSSM_DATA_PTR  OutMessage);
    CSSM_RETURN (CSSMAPI *Capture)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_BIR_PURPOSE  Purpose,
        CSSM_HRS_BIR_HANDLE_PTR  CapturedBIR,
        sint32  Timeout,
        CSSM_HRS_BIR_HANDLE_PTR  AuditData);
    CSSM_RETURN (CSSMAPI *CreateTemplate)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_BIR_PURPOSE  Purpose,
        const CSSM_HRS_INPUT_BIR *CapturedBIR,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR  NewTemplate,
        const CSSM_DATA *Payload);
    CSSM_RETURN (CSSMAPI *Process)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const CSSM_HRS_INPUT_BIR *CapturedBIR,
        CSSM_HRS_BIR_HANDLE_PTR  ProcessedBIR);
    CSSM_RETURN (CSSMAPI *VerifyMatch)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *ProcessedBIR,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE *AdaptedBIR,
        CSSM_BOOL *Result,
        CSSM_HRS_FAR_PTR  FARAchieved,
        CSSM_HRS_FRR_PTR  FRRAchieved,
        CSSM_DATA_PTR *Payload);
    CSSM_RETURN (CSSMAPI *IdentifyMatch)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *ProcessedBIR,
        const CSSM_HRS_IDENTIFY_POPULATION *Population,
        CSSM_BOOL  Binning,
        uint32  MaxNumberOfResults,
        uint32 *NumberOfResults,
        CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
        sint32  Timeout);
    CSSM_RETURN (CSSMAPI *Enroll)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_BIR_PURPOSE  Purpose,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR  NewTemplate,
        const CSSM_DATA *Payload,
        sint32  Timeout,
        CSSM_HRS_BIR_HANDLE_PTR  AuditData);
    CSSM_RETURN (CSSMAPI *Verify)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR  AdaptedBIR,
        CSSM_BOOL *Result,
        CSSM_HRS_FAR_PTR  FARAchieved,
        CSSM_HRS_FRR_PTR  FRRAchieved,
        CSSM_DATA_PTR *Payload,
        sint32  Timeout,
        CSSM_HRS_BIR_HANDLE_PTR  AuditData);
    CSSM_RETURN (CSSMAPI *Identify)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_IDENTIFY_POPULATION *Population,
        CSSM_BOOL  Binning,
        uint32  MaxNumberOfResults,
        uint32 *NumberOfResults,
        CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
        sint32  Timeout,
        CSSM_HRS_BIR_HANDLE_PTR  AuditData);
    CSSM_RETURN (CSSMAPI *Import)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const CSSM_DATA *InputData,
        CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT  InputFormat,
        CSSM_HRS_BIR_PURPOSE  Purpose,
        CSSM_HRS_BIR_HANDLE_PTR  ConstructedBIR);
    CSSM_RETURN (CSSMAPI *SetPowerMode)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_POWER_MODE  PowerMode);
    CSSM_RETURN (CSMAPI *DbOpen)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const uint8 *DbName,
        CSSM_HRS_DB_ACCESS_TYPE  AccessRequest,
        CSSM_HRS_DB_HANDLE_PTR  DbHandle,
        CSSM_HRS_DB_CURSOR_PTR  Cursor);
    CSSM_RETURN (CSSMAPI *DbClose)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_DB_HANDLE_PTR  DbHandle);
    CSSM_RETURN (HRS *DbCreate)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const uint8 *DbName,
        CSSM_HRS_DB_ACCESS_TYPE  AccessRequest,
        CSSM_HRS_DB_HANDLE_PTR  DbHandle);
    CSSM_RETURN (CSSMAPI *DbDelete)
        (CSSM_HRS_HANDLE  ModuleHandle,
        const uint8  *DbName);
    CSSM_RETURN (CSSMAPI *DbSetCursor)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_DB_HANDLE  DbHandle,
        const CSSM_GUID *KeyValue,
        CSSM_HRS_DB_CURSOR_PTR  Cursor);
    CSSM_RETURN (CSSMAPI *DbFreeCursor)
        (CSSM_HRS_HANDLE  ModuleHandle,
        CSSM_HRS_DB_CURSOR_PTR  Cursor);
    CSSM_RETURN (CSSMAPI *DbStoreBIR)
```

```
            (CSSM_HRS_HANDLE  ModuleHandle,
            const CSSM_HRS_INPUT_BIR *BIRToStore,
            CSSM_HRS_DB_HANDLE  DbHandle,
            CSSM_GUID_PTR  Guid);
        CSSM_RETURN (CSSMAPI *DbGetBIR)
            (CSSM_HRS_HANDLE  ModuleHandle,
            CSSM_HRS_DB_HANDLE  DbHandle,
            const CSSM_GUID *KeyValue,
            CSSM_HRS_BIR_HANDLE_PTR  RetrievedBIR,
            CSSM_HRS_DB_CURSOR_PTR  Cursor);
        CSSM_RETURN (CSSMAPI *DbGetNextBIR)
            (CSSM_HRS_HANDLE  ModuleHandle,
            CSSM_HRS_DB_CURSOR_PTR  Cursor,
            CSSM_HRS_BIR_HANDLE_PTR  RetrievedBIR,
            CSSM_GUID_PTR  Guid);
        CSSM_RETURN (CSSMAPI *DbQueryBIR)
            (CSSM_HRS_HANDLE  ModuleHandle,
            CSSM_HRS_DB_HANDLE  DbHandle,
            const CSSM_HRS_INPUT_BIR *BIRToQuery,
            CSSM_GUID_PTR  Guid);
        CSSM_RETURN (CSSMAPI *DbDeleteBIR)
            (CSSM_HRS_HANDLE  ModuleHandle,
            CSSM_HRS_DB_HANDLE  DbHandle,
            const CSSM_GUID *KeyValue);
    } CSSM_SPI_HRS_FUNCS, *CSSM_SPI_HRS_FUNCS_PTR;
```

3.2 SPI Definitions

To avoid unnecessary duplication of function call definitions, the SPI function call definitions are included with their corresponding API function call definitions in Chapter 2 (on page 11).

Appendix A

Conformance

A.1 Status of this Appendix

Claims to conformance on any aspect of the entities identified in this Technical Standard can only be made in relation to associated Open Group definitions of *products*, as specified in associated Open Group *Product Standards* and their related *Conformance Statement Questionnaires*.

In this regard, the content of this Appendix is advisory only, for guidance purposes as to the intentions of the HRS API designers.

A.2 Design Concepts for Conformance

Conformance to the HRS specification is envisioned as falling into the following two categories:

1. An HRS-compliant application
2. An HRS-compliant service provider

A.3 HRS-Compliant Application

It is envisioned that to claim compliance as an HRS application, a software application should, for each HRS function call utilized, perform that operation consistent with the specification. That is, all input parameters should be present and valid.

All HRS function calls should be available. It is not intended that there should be a minimum set of functions that might be called.

A.4 HRS-Compliant Service Providers

It is envisioned that to claim compliance as an HRS service provider, a service provider should implement mandatory functions for their category, defined below, in accordance with the SPI defined in Chapter 3 (on page 75).

Service Providers are categorized as either Verification SPs or Identification SPs. In addition, an HRS-SP may be categorized as a monolithic or explicit client/server SP. A monolithic HRS-SP is fully loaded on a single platform. A client/server HRS-SP has components installed on two or more platforms. These components (client and server) may communicate with each other using the streaming callbacks provided by the client/server application.

HRS-SPs should accept all valid input parameters and return valid outputs. Optional capabilities and returns are not required to claim conformance. However, any optional functions or parameters that are implemented should be implemented in accordance with the specification requirements.

Additionally, all HRS-SPs should provide all required module registry entries. Entries to the module registry should be performed upon installation.

HRS-SPs should possess a valid and unique GUID, which may be self-generated.

Biometric data generated by the HRS-SP should conform to the data structures defined in Section 2.1 (on page 11). HRS-SPs may only return CSSM_HRS_BIR data containing a registered *FormatOwner* with an associated valid *FormatType*.

All BSPs should support all *Handle* operations; see Section 2.4.1 (on page 28). Database operations (see Section 2.4.4 (on page 62)) are optional.

The following table is a summary of HRS-SP conformance requirements by type. Details are provided in the following sections.

Function	Verification SP	Identification SP
Handle Functions		
HRS_FreeBIRHandle	X	X
HRS_GetBIRFromHandle	X	X
HRS_GetHeaderFromHandle	X	X
Callback and Event Functions		
HRS_EnableEvents		
HRS_SetGUICallbacks		
HRS_SetStreamCallback		
HRS_StreamInputOutput		
Biometric Functions		
HRS_Capture		
HRS_CreateTemplate		
HRS_Process		
HRS_VerifyMatch		
HRS_IdentifyMatch		
HRS_Enroll	X	X
HRS_Verify	X	X
HRS_Identify		X
HRS_Import		
HRS_SetPowerMode		
Database Functions		
HRS_DbOpen		
HRS_DbClose		
HRS_DbCreate		
HRS_DbDelete		
HRS_DbSetCursor		
HRS_DbFreeCursor		
HRS_DbStoreBIR		
HRS_DbGetBIR		
HRS_DbGetNextBIR		
HRS_DbQueryBIR		
HRS_DbDeleteBIR		

Table A-1 HRS-SP Conformance Requirements by Type

A.4.1 HRS-Compliant Verification SPs

Verification SPs are those which are capable of performing 1:1 matching (or authentication), but not 1:N identification matching.

Note: 1:few matching may be supported as a series of 1:1 calls.

An HRS-compliant Verification SP should support the following biometric functions:

HRS_Enroll() Only *Purpose* flags indicating *Enroll for Verification* should be accepted. If another purpose is set, an error condition should be set as a CSSM_RETURN. Acceptance of *Payload* is optional.

Client/server implementations of this function are optional.

HRS_Verify( ) Client/server implementations of this function are optional.

Only the nearest, better supported *RequestedFAR* should be supported; however, the SP should return that supported value (*ActualFAR*).

Return of payload is required only if one is contained within the input *StoredTemplate* and if the score sufficiently exceeds the *ActualFAR*.

Return of *AdaptedBIR* is optional. Return of raw data (*AuditData*) is optional.

As a default, all SPs should provide any GUI associated with the capture portion of the *Verify* operation. However, support for application control of the GUI is optional.

A.4.2 HRS-Compliant Identification SPs

Identification BSPs are those which are capable of performing both 1:N identification matching as well as 1:1 matching (or authentication). An HRS-compliant *Identification* BSP should support the following biometric functions:

HRS_Enroll( ) *Purpose* flags indicating either *Enroll for Verification* or *Enroll for Identification* should be accepted.

Acceptance of *Payload* is optional.

Client/server implementations of this function are optional.

HRS_Verify( ) Client/server implementations of this function are optional.

Only the nearest, better supported *RequestedFAR* should be supported; however, the BSP should return that supported value (*ActualFAR*).

Return of payload is required only if one is contained within the input *StoredTemplate* and if the score sufficiently exceeds the *ActualFAR*.

Return of *AdaptedBIR* is optional. Return of raw data (*AuditData*) is optional.

As a default, all SPs should provide any GUI associated with the capture portion of the *Verify* operation. However, support for application control of the GUI is optional.

HRS_Identify( ) Client/server implementations of this function are optional.

Only the nearest, better supported *RequestedFAR* should be supported; however, the BSP should return that supported value (*ActualFAR*).

Support of *binning* is optional.

Return of matching *Candidates* is required; however, the SP may return values for the *ActualFAR* field as next nearest step / increment.

As a default, all SPs should provide any GUI associated with the capture portion of the *Identify* operation. However, support for application control of the GUI is optional.

A.4.3 HRS-Compliant Client/Server SPs

An HRS-SP instantiation may be wholly installable and loaded onto a single platform. This is referred to as a ‘‘Local’’ HRS-SP. However, an SP may be instantiated and loadable as separate client / server components which operate and communicate together. This is referred to as a ‘‘Distributed’’ HRS-SP.

If an HRS-SP is constructed such that it is installed and operates in a distributed fashion across a network or other communications channel, with direct communication between the distributed HRS components, it is considered to be a Distributed (Client / Server) HRS service provider. This does not include a Local service provider that can be installed in whole on both a client and a server platform, where communication between the client and the server is always at the application layer and the two instantiations of the service provider do not communicate with each other directly.

HRS-SPs should post to the module registry whether they support Local operation, Distributed operation, or both.

Local HRS-SPs should support all functions required by their category. These functions are both called locally (by an application running on the same platform) and executed locally (on the platform on which they are called and on which the service provider is installed). Streaming callbacks are not supported or used by local service providers.

In order to be HRS-compliant, Client / Server service providers should support all functions for their category (Verification or Identification) defined above, when initiated from an application on either side. That is, all functions are supported by both the client and server components. However, although called locally, some functions may be executed remotely. These are identified below.

Executed Locally	Executed Locally or Remotely
HRS_Capture*	HRS_Enroll
HRS_CreateTemplate*	HRS_Verify
HRS_Process*	HRS_Identify
HRS_VerifyMatch*	
HRS_IdentifyMatch*	
HRS_Import*	

Table A-2 Function Calls Executed Locally / Remotely

Local functions are always executed on the system where the API is called. Eligible functions are executed remotely if a *Streaming Callback* has been set.

* Optional function.

Additionally, Distributed HRS-SPs should support direct communication between *partner* components by ‘‘tunneling’’ through the application layer using the streaming callback capability (via the *HRS_SetStreamCallback*( ) and *HRS_StreamInputOutput*( ) functions).

A.4.4 Optional Capabilities

The following capabilities are considered optional in terms of HRS support and compliance. Note that:

- If implemented, optional capabilities should conform to specification definitions.
- HRS-SPs are required to post to the module registry whether or not each option is supported.
- HRS-SP documentation should include a table that identifies which options are supported and which options are not supported.

A.4.4.1 Optional Functions

Primitive Functions

Although it was originally intended that the primitive functions (CSSM_HRS_Capture, CSSM_HRS_CreateTemplate, CSSM_HRS_Process, CSSM_HRS_VerifyMatch, and CSSM_HRS_IdentifyMatch) would be mandatory, it was decided that this would place undue burden on manufacturers of ‘‘self-contained devices’’ in which the biometric processing/matching is performed within the device itself. Therefore, these functions have not been specified as required for HRS service providers to be considered ‘‘HRS-compliant’’. However, it is highly recommended that, if supported by the underlying technology, these functions be included in the service provider.

CSSM_HRS_Capture( )

If this function is supported, Verification BSPs need only accept *Purpose* flags indicating *Verification* or *Enroll* for verification. If another purpose is set by the application, an error condition may be set as a CSSM_RETURN. Similarly, this function need only return *CapturedBIR* with the *Purpose* mask set to *Verification* or *Enrol* for verification. If this function is supported, *Identification* service providers must accept all possible *Purpose* flags (except *Audit*), even if there is no difference in the content or format of the returned data.

Return of raw data (ac *AuditData*) is optional.

As a default, all service providers must provide any GUI associated with the *Capture* operation. However, support for application control of the GUI is optional.

CSSM_HRS_CreateTemplate( )

If this function is supported, verification service providers need only accept input *CapturedBIR*s with the *Purpose* set to *Enroll* for verification. f another purpose is set, an error condition may be set as a CSSM_RETURN.

If this function is supported, identification service providers must accept input *CapturedBIR*s with both *Enroll* purposes.

Acceptance of Payload is optional.

Adaptation of an existing template is optional.

CSSM_HRS_Process( )

If this function is supported, verification service providers need only accept input *CapturedBIR*s with the *Purpose* set to verification. If another purpose is set, an error condition may be set as a CSSM_RETURN.

If this function is supported, identification service providers must accept input *CapturedBIR*s with all possible *Purpose* flags (except *Audit*), even if there is no difference in the content or format of the returned data.

CSSM_HRS_VerifyMatch( )

If this function is supported, only input BIRs (*ProcessedBIR*) with the *Purpose* mask including a value of *Verification*, and (*StoredTemplate*) with the *Purpose* mask including a value of *Enroll* for verification must be accepted. If another *Purpose* is set, an error condition may be set as a CSSM_RETURN.

Only the nearest, better supported *RequestedFAR* must be supported; however, the service provider must return that supported value (*ActualFAR*).

Return of payload is required only if one is contained within the input *StoredTemplate* and if the score is better than the *ActualFAR* (the service provider must post the minimum FAR required to return payload in the module registry).

Return of AdaptedBIR is optional.

BioAPI_IdentifyMatch( )

May be supported by identification service providers for BIRs whose *purpose* is *Identify*.

Only the nearest, better supported *RequestedFAR* must be supported; however, the service provider must return that supported value (*ActualFAR*).

Support of binning is optional.

Return of matching *Candidate*s is required; however, the service provider may return values for the *Score* field as next nearest step / increment.

Database Operations

HRS-SPs are not required to provide an internal (or internally controlled) database. If one is provided, however, *all* database functions should be provided in order to access and maintain it. All provided database functions should conform to the definitions.

HRS_Import( )

This function, as a whole, is optional. If it is provided, the module registry should so reflect, and it should be implemented as defined.

Verification BSPs are only required to process imported data if the *Purpose* flag includes *Enroll* or *Enroll for Verification*.

HRS_SetPowerMode( )

This function, as a whole, is optional. If it is provided, the module registry should so reflect, and it should be implemented as defined.

Application-Controlled GUI

The SP supports the necessary callbacks to allow the application to control the look-and-feel of the GUI.

All HRS-compliant SPs should provide, as a default, the capability to display user interface information (assuming such a display is present and required by the authentication technology).

All SP-supplied GUIs should include an operator abort/cancel mechanism.

Optionally, the HRS-SP may also support application-controlled GUI. In this case, the SP should support the *HRS_SetGUICallbacks*( ) function.

If application-controlled GUI is supported, the HRS-SP may additionally provide streaming data (for example, for voice and face recognition). If streaming data is provided, the SP should support the input parameters *GuiStreamingCallback* and *GuiStreamingCallbackCtx*.

All HRS-SPs should post to the module registry the GUI functions/options it supports.

A.4.4.2 Optional Sub-Functions

The following table identifies optional capabilities, each of which the HRS-SP should declare (by filling in the table) as being supported or not supported (both in the module registry and in the SP documentation).

Capability	Supported	Not Supported
Return of raw/audit data		
Return of quality		
Application-controlled GUI		
GUI streaming callbacks		
Detection of source presence		
Payload carry		
BIR signing		
BIR encryption		
Return of FRR		
Model adaptation		
Binning		
Client/server communications		
Supports self-contained device		

Table A-3 Support for HRS-SP Optional Sub-Functions

Return of Raw Data

Functions involving the capture of biometric data from a sensor may optionally support the return of this raw data for purposes of display or audit. If supported, the output parameter *AuditData* contains a pointer to this data (or CSSM_HRS_INVALID_BIR_HANDLE to indicate that no audit data is available). If not supported, the SP returns a value of –1.

Return of Quality

Upon the new capture of biometric data from a sensor, the SP may calculate a relative quality value associated with this data, which it will include in the header of the returned *CapturedBIR* (and the optional *AuditData*). If supported, this header field will be filled with a positive value between 1 and 100. If not supported, this field will be set to –2. This would occur during *HRS_Capture*( ) and *HRS_Enroll*( ).

Similarly, when a BIR is processed, another quality calculation may be performed and the quality value included in the header of the *ProcessedBIR* (and the optional *AdaptedBIR*). This would occur during the following *ConstructedBIR* operations:

HRS_CreateTemplate( )
HRS_Process( )
HRS_Verify( )
HRS_VerifyMatch( )
HRS_Enroll( )
HRS_Import( )

The HRS-SP should post to the module registry whether or not it supports the calculation of quality measurements for each type of BIR—raw, intermediate, and processed.

Application-Controlled GUI

The HRS-SP supports the necessary callbacks to allow the application to control the ‘‘look-and-feel’’ of the GUI.

GUI Streaming Callbacks

The HRS-SP provides GUI streaming data, and supports the GUI streaming callback.

Detection of Source Presence

The HRS-SP can detect when there are samples available, and supports the source present event.

Payload Carry

Some HRS-SPs may optionally support the encapsulation of a *Payload* within a processed BIR, and the subsequent release of that payload upon successful verification. The content of the payload is unrestricted, but may include secrets such as PINs or private keys associated with the user whose biometric data is contained within the BIR. Payload data is encapsulated during an *HRS_Enroll*( ) or *HRS_CreateTemplate*( ) operation, and is released as a result of a *HRS_Verify*( ) or *HRS_VerifyMatch*( ) operation. If supported, HRS-SPs should post to the module registry the maximum payload size that can be accommodated. A maximum size of zero indicates *payload carry* is not supported. If input payloads exceed this size, an error should be generated. If *payload carry* is not supported, the output Payload is set to a NULL pointer.

The HRS-SP should post to the module registry whether or not it provides the following:

- BIR encryption

- BIR signing

Return of FRR

During matching operations, the application may request a specific FAR threshold and the HRS-SP should return the nearest supported threshold. Optionally, the SP may also return the estimated FRR associated with this supported FAR. This Corresponding FRR value is an optional return from the *HRS_Verify*( ), *HRS_VerifyMatch*( ), *HRS_Identify*( ), and *HRS_IdentifyMatch*( ) functions. If supported, the HRS-SP will return its best estimate of expected FRR. If not supported, the SP will return a value of CSSM_HRS_FRR_NOT_SUPPORTED.

Template/Model Adaptation

Some HRS-SPs may optionally provide the capability to utilize newly captured biometric data to update a stored BIR. This may only be performed as a result of a successful *HRS_Verify*( ) or *HRS_VerifyMatch*( ) (that is, one in which *Result* = *CSSM_TRUE*). This is performed in order to keep the enrolled BIR as fresh as possible, with the highest possible quality. The SP makes the decision as to when and if the adaptation should be performed (based on such factors as quality, elapsed time, and significant differences). If the SP does not support adaptation, the returned *AdaptedBIR* is set to CSSM_HRS_UNSUPPORTED_BIR_HANDLE. If the SP supports adaptation, but for some reason is not able or chooses not to perform the adaptation, then the return *AdaptedBIR* is set to CSSM_HRS_INVALID_BIR_HANDLE. The SP should post to the module registry its support for adaptation.

Binning

Identification SPs may optionally support methods of limiting the population of a database to be searched in order to improve response time. This applies to *Identification* type SPs only and occurs only during *HRS_Identify*( ) and *HRS_IdentifyMatch*( ) operations. *Identification* SPs should post whether or not they support binning. *Identification* SPs that do not support binning may ignore the input Binning *on/off* parameter. Note that no explicit API support is provided for setting or modifying the binning strategy other than for turning it on or off.

Client/Server Communication

The SP supports the streaming callback to allow the client and server SPs to communicate.

Self-Contained Device

The supported device is self-contained; that is, at least the verification function is entirely contained within the device, and the device may contain a database.

Glossary

API

Application Programming Interface

BIR

Biometric Identification Record

BSP

Biometric Service Provider

CDSA

Common Data Security Architecture

CSSM

Common Security Services Manager: the infrastructure parts of the CDSA.

EMM

Elective Module Manager: an extensibility mechanism in CDSA supporting the dynamic addition of new categories of service, beyond the basic set of Cryptographic Service Provider (CSP), Trust Policy (TP), Authorization Computation (AC), Certificate Library (CL), and Data Storage Library (DL).

FAR

False Accept Rate: the probability that biometric data samples are falsely decided by the HRS as matching; that is, they should not match, but do.

FRR

False Reject Rate: the probability that biometric data samples are falsely decided by the HRS as not matching; that is, they should match, but do not.

GUI

Graphical User Interface

HRS

Human Recognition Service

IBIA

International Biometric Industry Association

MDS

Module Directory Service: a platform-independent registry service used by CDSA to name and locate software components and their security credentials.

Payload

Data wrapped inside biometric data for release to an application on successful verification of authenticity of a user. This can be any data that is useful to an application.

PIN

Personal Identification Number

SPI

Service Provider Interface

Index