



SecuAPI® Reference Manual

V1.05

The Fingerprint Biometric API Specification from SecuGen

SG1-0022A-003 (06/09)

© Copyright 1998-2009 SecuGen Corporation.

ALL RIGHTS RESERVED. Specifications are subject to change without notice. SecuGen and SecuAPI are registered trademarks of SecuGen Corporation. All other brands or products may be trademarks, service marks or registered trademarks of their respective owners.

Contents

BEFORE YOU BEGIN.....	V
BIOMETRICS OVERVIEW	V
ABOUT SECUGEN	V
ABOUT SECUGEN PRODUCTS	VI
CHAPTER 1. SECUAPI® OVERVIEW.....	7
1.1. PURPOSE	7
1.2. API MODEL.....	7
Low-Level APIs	7
High-level APIs	8
1.3. FIR	8
1.3.1 FIR Structure.....	8
1.4. PAYLOAD.....	9
CHAPTER 2. DATA STRUCTURES.....	10
2.1. BASIC TYPE DEFINITIONS	10
SecuAPI.....	10
SecuAPI_SINT8.....	10
SecuAPI_SINT16.....	10
SecuAPI_SINT32.....	10
SecuAPI_UINT8	10
SecuAPI_UINT16	10
SecuAPI_UINT32	10
SecuAPI_VOID_PTR.....	11
SecuAPI_BOOL.....	11
SecuAPI_CHAR.....	11
SecuAPI_CHAR_PTR.....	11
SecuAPI_TRUE.....	11
SecuAPI_FALSE.....	11
SecuAPI_NULL.....	11
2.2. DATA STRUCTURES & TYPE DEFINITIONS	11
SecuAPI_AUDIT_DATA	11
SecuAPI_DEVICE_ID.....	12
SecuAPI_DEVICE_INFO_0.....	12
SecuAPI_DEVICE_INFO_1.....	12
SecuAPI_DEVICE_INFO_PTR.....	13
SecuAPI_DEVICE_NAME.....	13
SecuAPI_EXPORT_DATA	13
SecuAPI_EXPORT_AUDIT_DATA	14
SecuAPI_EVENT.....	14
SecuAPI_FINGER_DATA	14
SecuAPI_FIR	15
SecuAPI_FIR_DATA	15
SecuAPI_FIR_DATA_TYPE.....	15
SecuAPI_FIR_FORMAT	15
SecuAPI_FIR_HEADER.....	16
SecuAPI_FIR_HANDLE.....	16
SecuAPI_FIR_PAYLOAD.....	16
SecuAPI_FIR_PURPOSE.....	16
SecuAPI_FIR_QUALITY.....	17
SecuAPI_FIR_FORMAT	17

MINCONV_DATA_TYPE	17
SecuAPI_FIR_SECURITY_LEVEL	17
SecuAPI_FIR_TEXTENCODING	18
SecuAPI_FIR_VERSION	18
SecuAPI_HANDLE	18
SecuAPI_HWND	18
SecuAPI_IMAGE_DATA	18
SecuAPI_INIT_INFO_0	19
SecuAPI_INIT_INFO_PTR	19
SecuAPI_INPUT_FIR	19
SecuAPI_INPUT_FIR_FORM	20
SecuAPI_MAX_DEVICE	20
SecuAPI_RETURN	20
SecuAPI_TEMPLATE_DATA	20
SecuAPI_VERSION	20
SecuAPI_WINDOW_CALLBACK	20
SecuAPI_WINDOW_CALLBACK_INFO	20
SecuAPI_WINDOW_CALLBACK_PARAM_0	21
SecuAPI_WINDOW_CALLBACK_PARAM_1	21
SecuAPI_WINDOW_CALLBACK_PARAM_PTR	21
SecuAPI_WINDOW_STYLE	21
SecuAPI_WINDOW_OPTION	21
2.3. ERROR CONSTANTS	23
CHAPTER 3. FUNCTIONS.....	25
3.1. BASIC FUNCTIONS	25
SecuAPI_Init	25
SecuAPI_Terminate	25
SecuAPI_GetVersion	25
SecuAPI_GetInitInfo	26
SecuAPI_SetInitInfo	26
SecuAPI_EnumerateDevice	27
SecuAPI_GetDeviceInfo	27
SecuAPI_SetDeviceInfo	28
SecuAPI_OpenDevice	29
SecuAPI_CloseDevice	29
SecuAPI_AdjustDevice	29
SecuAPI_MonitorDevice	30
3.2. MEMORY FUNCTIONS	30
SecuAPI_FreeFIRHandle	30
SecuAPI_GetFIRFromHandle	31
SecuAPI_GetHeaderFromHandle	31
SecuAPI_GetExtendedFIRFromHandle	32
SecuAPI_GetExtendedHeaderFromHandle	32
SecuAPI_FreeFIR	33
SecuAPI_GetTextFIRFromHandle	33
SecuAPI_GetExtendedTextFIRFromHandle	34
SecuAPI_FreeTextFIR	35
SecuAPI_FreePayload	35
SecuAPI_FreeExportData	35
SecuAPI_FreeExportAuditData	36
3.3. BSP FUNCTIONS	36
SecuAPI_Capture	36
SecuAPI_Process	37
SecuAPI_CreateTemplate	37

<i>SecuAPI_VerifyMatch</i>	38
<i>SecuAPI_Enroll</i>	39
<i>SecuAPI_Verify</i>	40
3.4. USER INTERFACE FUNCTIONS	41
<i>SecuAPI_SetSkinResource</i>	41
3.5. UTILITY FUNCTIONS	41
<i>SecuAPI_FDxToSecuBSP</i>	41
<i>SecuAPI_SecuBSPToFDx</i>	42
<i>SecuAPI_ExportImage</i>	42

Before You Begin

Biometrics Overview

Biometrics is an automated method of recognizing a person based on physical or behavioral characteristics. Biometric information that can be used to accurately identify people includes fingerprint, voice, face, iris, handwriting and hand geometry.

There are two key functions offered by a biometric system. One method is **identification**, a “one-to-many” matching process in which a biometric sample is compared sequentially to a set of stored samples to determine the closest match. The other is **verification**, a “one-to-one” matching process in which the biometric system checks previously enrolled data for a specific user to verify whether that individual is who he or she claims to be. The verification method provides the best combination of speed and security, especially where multiple users are concerned, and requires a user ID or other identifier for direct matching.

With an increasing reliance on online technology and other shared resources, the information age is quickly revolutionizing the way transactions are initiated and completed. Business transactions of all types are increasingly being handled online and remotely. This unprecedented growth in electronic transactions has underlined the need for a faster, more secure and more convenient method of user verification than passwords can provide.

Using biometric identifiers offers several advantages over traditional and current methods. This is because only biometric authentication is based on the identification of an intrinsic part of a human being. Tokens such as smart cards, magnetic stripe cards and physical keys, can be lost, stolen, duplicated or left behind. Passwords can be forgotten, shared, hacked or unintentionally observed by a third party. By eliminating all of these potential trouble spots, biometric technology can provide greater security, with convenience, needed for today's complex electronic landscape.

Advantages of Using Fingerprints

The advantages of using fingerprints include widespread public acceptance, convenience and reliability. It takes little time and effort to scan one's fingerprint with a fingerprint reader, and so fingerprint recognition is considered among the least intrusive of all biometric verification techniques. Ancient officials used thumbprints to seal documents thousands of years ago, and law enforcement agencies have been using fingerprint identification since the late 1800s. Fingerprints have been used so extensively and for so long, there is a great accumulation of scientific data supporting the idea that no two fingerprints are alike.

About SecuGen

SecuGen (www.secugen.com) provides biometric solutions for physical and network security employing advanced fingerprint recognition technology. The company's comprehensive product line includes high quality optical fingerprint readers and sensor component, software and development kits that are used for a variety of innovative applications including Internet, enterprise network and desktop security, physical access control, time and attendance management and financial and medical records control. SecuGen patented products feature the industry's longest warranty and are renowned for their accuracy, reliability and versatility. Based in Silicon Valley, SecuGen has been serving the global biometric community since 1998 and is an active member of the Biometrics Consortium (www.biometrics.org), the BioAPI Consortium (www.bioapi.org) and the International Biometric Industry Association (www.ibia.org).

About SecuGen Products

SecuGen Sensor Qualities

- **Excellent Image Quality:** Clear, distortion-free fingerprint images are generated using advanced, patent-pending optical methods. Quality imaging yields better sampling for minutiae data extraction.
- **Durability:** Mechanical strength tests show resistance to impact, shock and scratches.
- **Powerful Software:** Precise, fast processing algorithm ensures efficiency and reliability.
- **Ruggedness and Versatility:** Solid engineering and superior materials allows for use under extreme conditions.
- **Ergonomic Design:** Compact, modular design for seamless integration into small devices, ease of use, and compatibility make it ideal for a broad range of applications.
- **Low Cost:** Products are developed to deliver high performance, zero maintenance at very affordable prices for general and industrial use.

Advantages of SecuGen Sensors Over Other Optical Sensors

- Unique optical method captures fine details, even from dry skin
- Extremely low image-distortion
- Reinforced materials
- Wear resistance
- Attractively small size
- Ease of integration
- Ready-to-use
- Low cost through longer life and no maintenance requirements

Advantages SecuGen Sensors Over Semiconductor (Capacitive) Sensors

- Non-metal, non-silicon components make it less susceptible to corrosion when exposed to salts, oil and moisture from skin and environment
- Superior surface properties eliminate need for costly coating and processing procedures
- Greater mechanical strength, wear-resistance, and durability
- Broader range of applicability, especially for use in extreme conditions and climates
- Immunity from electrostatic discharge
- Low cost through longer life and no maintenance requirements

Strengths of SecuGen Software and Algorithms

- Unique image processing algorithm extracts fingerprint minutiae very accurately
- High signal-to-noise ratio processing algorithm screens out false features
- Highly efficient matching algorithm
- Fast overall process of extraction, matching and verification
- Encryption function to protect user privacy
- Compatibility with existing desktop, laptop PCs interface computers
- Ease in developing applications for various purposes

CHAPTER 1. SecuAPI® Overview

1.1. Purpose

SecuAPI® is SecuGen's fingerprint biometric application programming interface (API), which encapsulates SecuGen's fingerprint authentication technology in a well-designed, feature-rich biometric authentication model.

SecuAPI provides not only the basic functions needed for any biometric system - enrollment and verification, but also the low-level functions that can be used in client/server environments. With these functions combined, it is possible for applications to manage the capture of biometric samples on a client, while the enrollment or verification processes are carried out on a server.

Designed for easy integration into the industry standard BioAPI™ framework, SecuAPI supports the existing set of BioAPI features and adds new, useful functions such as obtaining device information, among others. **Note:** This version of SecuAPI does not support one-to-many (1:N) matching.

1.2. API Model

SecuAPI includes both low- and high-level functions for the greatest flexibility in programming. In stand-alone applications (i.e. in non-client/server environments), developers will typically use the high-level APIs.

For some CPU-intensive applications, however, the processing of fingerprint data may need to be accomplished in multiple stages, in which cases, the low-level functions would be implemented to perform both capture functions (to capture of raw fingerprint samples) and matching functions (to match input samples with stored templates).

A typical example of this multi-stage approach can be found in a client/server environment where fingerprint templates are captured at the client, but matching takes place on the server. The SecuAPI low-level functions provide the additional, granular control that may be needed within complex applications.

Low-Level APIs

Capture

The Capture function captures a fingerprint image from a SecuGen fingerprint reader, extracts feature points (minutiae) from the image, and returns a Fingerprint Identification Record (FIR) as the result. Multiple samples are captured for the purposes of enrollment, verification, or identification. The application specifies the purpose for which the samples are intended, i.e. enrollment, verification or identification, and records the purpose within the constructed FIR.

The Capture function may process the fingerprint data for as long as necessary. If processing is incomplete, the Capture function returns an "intermediate" Fingerprint Identification Record (**FIR**), indicating that the Process function needs to be called. Otherwise, if processing is complete, the Capture function returns a "processed" FIR, indicating that the Process function does not need to be called. (See also section 1.3, FIR, for more information)

Process

The Process function processes fingerprint data samples for the purposes of verifying or identifying users; it is not used for enrollment (registration) purposes. The Process function accepts both raw and intermediate FIRs as input and returns as the result, intermediate or processed FIRs, respectively.

VerifyMatch

The VerifyMatch function compares a newly input FIR with a stored FIR and returns the result of the comparison.

CreateTemplate

The CreateTemplate function processes samples for the construction of an *enrollment FIR*. CreateTemplate can take an intermediate or processed FIR from the Capture function as its input. Optionally, CreateTemplate can use an old FIR to construct a new FIR. The application can also provide payload to wrap inside the new template.

High-level APIs**Enroll**

The Enroll function extracts feature points (minutiae) from captured fingerprint images using SecuGen fingerprint readers. Optionally, Enroll can use an old FIR to create a new FIR. The application can also provide payload to wrap inside the new template.

Verify

The Verify function captures a fingerprint image from a SecuGen fingerprint reader, extracts feature points (minutiae) from the image, compares the extracted data against a stored FIR, and returns the result of the comparison. If matching is successful and payload is in the stored template, the Verify function will also return payload to the application.

1.3. FIR

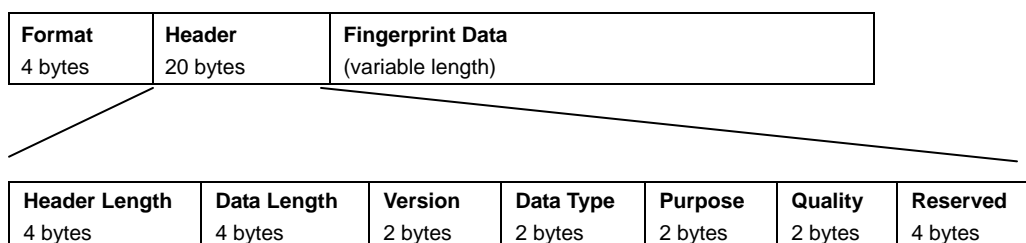
The Fingerprint Identification Record (FIR) is used to store fingerprint data and associated information that are returned to the SecuAPI-based application following a requested operation. The FIR may include raw, intermediate, or processed fingerprint data. **Note:** To match FIRs, the FIRs must contain processed fingerprint data.

- **Raw data** is simply the fingerprint image data
- **Intermediate data** is partially processed fingerprint data
- **Processed data** is the fully-processed fingerprint data

Note: The FIR may be generically referred to as a template. However, the term FIR more accurately distinguishes itself from other types of templates, such as that used in the FDx SDK (i.e. the 400 byte type of template). The SecuBSP SDK uses the FIR type of template.

1.3.1 FIR Structure

The FIR is composed of three fields: format, header, and opaque fingerprint data, as shown in the following diagram.

**Format**

The Format field indicates the format of the Fingerprint Data stored in the FIR and is 4 bytes in length. **Note:** If FIR format is changed, then the Header or Fingerprint Data structure can be changed. In this version, only SecuAPI_FIR_FORMAT_STANDARD is valid.

Header

The Header field contains information to process Fingerprint Data, is 20 bytes in length, and is comprised of seven sub-fields.

Header Length

Indicates length of the Header (bytes)

Data Length

Indicates length of the fingerprint data in the FIR (bytes)

Version

Indicates the FIR version number

Data Type

Indicates the type of fingerprint data stored in the FIR (e.g. raw, intermediate, or processed)

Purpose

Specifies the purpose of the FIR (e.g. enrollment, verification, or identification)

Quality

Indicates the quality of the fingerprint data on a scale of 1 to 100, where 1 is lowest and 100 is highest

Reserved

(This field is reserved for later use)

Fingerprint Data

The Fingerprint Data field contains raw, intermediate, or processed fingerprint data and is of variable length as determined by the values in the Format field. Fingerprint data size is stored in the Data Length sub-field of the Header.

1.4. Payload

Payload is any data used by an application, such as a user ID, password or cryptographic key, which is stored within a FIR and released to the application after successful verification.

During the enrollment process, an application may present a payload to be carried into the opaque data of the FIR that is being constructed. This payload is wrapped inside the biometric data by the **CreateTemplate** and **Enroll** functions.

For example, fingerprint samples themselves cannot be used as cryptographic keys because no two fingerprint samples captured from the same finger may be identical in all aspects. However, traditional cryptographic keys may be used instead by binding them closely to fingerprint data. SecuAPI allows the keys to be wrapped and stored inside the fingerprint data so that only a successful authentication by fingerprints can release the keys.

CHAPTER 2. Data Structures

2.1. Basic Type Definitions

SecuAPI

Definition of the SecuAPI calling convention

```
#ifdef (WIN32)
#define SecuAPI __stdcall
#else
#define SecuAPI
#endif
```

SecuAPI_SINT8

```
#ifdef WIN32
typedef __int8 SecuAPI_SINT8;
#endif
```

SecuAPI_SINT16

```
#ifdef WIN32
typedef __int16 SecuAPI_SINT16;
#endif
```

SecuAPI_SINT32

```
#ifdef WIN32
typedef __int32 SecuAPI_SINT32;
#endif
```

SecuAPI_UINT8

```
#ifdef WIN32
typedef BYTE SecuAPI_UINT8;
#endif
```

SecuAPI_UINT16

```
#ifdef WIN32
typedef WORD SecuAPI_UINT16;
#endif
```

SecuAPI_UINT32

```
#ifdef WIN32
typedef DWORD SecuAPI_UINT32;
```

```
#endif
```

SecuAPI_VOID_PTR

```
#ifdef WIN32
typedef void* SecuAPI_VOID_PTR;
#endif
```

SecuAPI_BOOL

```
#ifdef WIN32
typedef BOOL SecuAPI_BOOL;
#endif
```

SecuAPI_CHAR

```
#ifdef WIN32
typedef CHAR SecuAPI_CHAR;
#endif
```

SecuAPI_CHAR_PTR

```
#ifdef WIN32
typedef LPSTR SecuAPI_CHAR_PTR;
#endif
```

SecuAPI_TRUE

```
#define SecuAPI_FALSE (0)
```

SecuAPI_FALSE

```
#define SecuAPI_FALSE (SecuAPI_FALSE)
```

SecuAPI_NULL

```
#define SecuAPI_NULL (0)
```

2.2. Data Structures & Type Definitions

SecuAPI_AUDIT_DATA

Contains fingerprint image data and is used in SecuAPI_EXPORT_AUDIT_DATA

```
typedef struct secuapi_audit_data {
    SecuAPI_UINT32      Length;
    SecuAPI_UINT8      FingerID;
    SecuAPI_IMAGE_DATA_PTR Image;
} SecuAPI_AUDIT_DATA, *SecuAPI_AUDIT_DATA_PTR;
```

- **Members**

Length: Size of Structure.

FingerID: Represent finger position of each finger

```
#define SecuAPI_FINGER_ID_UNKNOWN      (0) // for verify
#define SecuAPI_FINGER_ID_RIGHT_THUMB (1)
#define SecuAPI_FINGER_ID_RIGHT_INDEX (2)
#define SecuAPI_FINGER_ID_RIGHT_MIDDLE (3)
#define SecuAPI_FINGER_ID_RIGHT_RING  (4)
#define SecuAPI_FINGER_ID_RIGHT_LITTLE (5)
#define SecuAPI_FINGER_ID_LEFT_THUMB  (6)
#define SecuAPI_FINGER_ID_LEFT_INDEX  (7)
#define SecuAPI_FINGER_ID_LEFT_MIDDLE (8)
#define SecuAPI_FINGER_ID_LEFT_RING   (9)
#define SecuAPI_FINGER_ID_LEFT_LITTLE (10)
```

Image: Pointer to SecuAPI_IMAGE_DATA structure. See SecuAPI_IMAGE_DATA structure

SecuAPI_DEVICE_ID

Consists of device instance and device name. The high order byte is the device instance; the low order byte is the device name.

```
typedef SecuAPI_UINT16      SecuAPI_DEVICE_ID
#define SecuAPI_DEVICE_ID_NONE (0x0000)
#define SecuAPI_DEVICE_ID_AUTO (0x00ff)
```

SecuAPI_DEVICE_INFO_0

Gets device information. Refer to GetDeviceInfo() and SetDeviceInfo().

```
typedef struct secuapi_device_info_0 {
    SecuAPI_UINT32    StructureType;    // must be 0
    SecuAPI_UINT32    ImageWidth;      // read only
    SecuAPI_UINT32    ImageHeight;     // read only
    SecuAPI_UINT32    Brightness;
    SecuAPI_UINT32    Contrast;
    SecuAPI_UINT32    Gain;
} SecuAPI_DEVICE_INFO_0, *SecuAPI_DEVICE_PTR_0;
```

- **Members**

StructureType:	Must be 0.
ImageWidth:	Image width in pixels (depends on device type)
ImageHeight:	Image height in pixels (depends on device type)
Brightness:	Brightness value of the device (ranges from 0 to 100)
Contrast:	Contrast value of the device (ranges from 0 to 100)
Gain:	Gain value of the device (depends on device type)

SecuAPI_DEVICE_INFO_1

Gets device information. Refer to GetDeviceInfo() and SetDeviceInfo().

```
typedef struct secuapi_device_info_1 {
    SecuAPI_UINT32    StructureType;    // must be 1
    SecuAPI_UINT32    ImageWidth;      // read only
    SecuAPI_UINT32    ImageHeight;     // read only
```

```

        SecuAPI_UINT32    Brightness;
        SecuAPI_UINT32    Contrast;
        SecuAPI_UINT32    Gain;
        SecuAPI_UINT32    ImageDPI;        // read only
        SecuAPI_UINT32    FWVersion;       // read only
        SecuAPI_UINT8     DeviceSN[16];    // read only
    } SecuAPI_DEVICE_INFO_1, *SecuAPI_DEVICE_INFO_PTR_1;

```

- **Members**

StructureType:	Must be 1.
ImageWidth:	Image width in pixels (depends on device type)
ImageHeight:	Image height in pixels (depends on device type)
Brightness:	Brightness value of the device (ranges from 0 to 100)
Contrast:	Contrast value of the device (ranges from 0 to 100)
Gain:	Gain value of the device (depends on device type)
ImageDPI:	Image resolution in DPI
FWVersion:	Firmware version
DeviceSN:	Device serial number

SecuAPI_DEVICE_INFO_PTR

```
typedef SecuAPI_VOID_PTR  SecuAPI_DEVICE_INFO_PTR
```

SecuAPI_DEVICE_NAME

```

typedef SecuAPI_UINT8          SecuAPI_DEVICE_NAME
#define SecuAPI_DEVICE_NAME_FDP02    (0x01)
#define SecuAPI_DEVICE_NAME_FDU01    (0x02)
#define SecuAPI_DEVICE_NAME_FDU02    (0x02)
#define SecuAPI_DEVICE_NAME_FDU03    (0x03)
#define SecuAPI_DEVICE_NAME_FDU04    (0x04)

```

Note: FDU03 includes FDU03- and SDU03-based readers.

SecuAPI_EXPORT_DATA

Contains exported minutiae data from FIR

```

typedef struct secuapi_export_data {
    SecuAPI_UINT32    Length;           /* sizeof of structure */
    SecuAPI_UINT8     EncryptType;      /* 0 = PC, 1 = FDA, */
    SecuAPI_UINT8     FingerNum;
    SecuAPI_UINT8     DefaultFingerID;  /* SecuAPI_FINGER_ID */
    SecuAPI_UINT8     SamplesPerFinger;
    SecuAPI_FINGER_DATA_PTR  FingerData;
    SecuAPI_UINT32     Reserved;
} SecuAPI_EXPORT_DATA, *SecuAPI_EXPORT_DATA_PTR;

```

- **Members**

Length:	Size of structure
EncryptType:	Encryption type: 0: PC (FDP/FDU), 1: FDA
FingerNum:	Number of fingers in FIR
DefaultFingerID:	Not used
SamplesPerFinger:	Number of samples per finger

FingerData: Pointer to SecuAPI_FINGER_DATA

SecuAPI_EXPORT_AUDIT_DATA

Contains exported image data from audit data

```
typedef struct secuapi_export_audit_data {
    SecuAPI_UINT32      Length;           /* sizeof of structure */
    SecuAPI_UINT8       FingerNum;
    SecuAPI_UINT8       SamplesPerFinger;
    SecuAPI_UINT32      ImageWidth;
    SecuAPI_UINT32      ImageHeight;
    SecuAPI_AUDIT_DATA_PTR AuditData;
    SecuAPI_UINT32      Reserved2;
} SecuAPI_EXPORT_AUDIT_DATA, *SecuAPI_EXPORT_AUDIT_DATA_PTR;
```

- **Members**

Length:	Size of structure
FingerNum:	Number of fingers in FIR
SamplesPerFinger:	Number of samples per finger
ImageWidth:	Image width in pixels
ImageHeight:	Image height in pixels
AuditData:	Pointer to SecuAPI_AUDIT_DATA structure. See SecuAPI_AUDIT_DATA structure

SecuAPI_EVENT

Defines event of SecuAPI

```
typedef SecuAPI_UINT16  SecuAPI_EVENT;

#define SecuAPI_WM_DEVICE_EVENT    0x8100    //SecuAPI Device event
#define SecuAPI_DEVICE_FINGER_OFF  0
#define SecuAPI_DEVICE_FINGER_ON   1
```

SecuAPI_FINGER_DATA

Contains finger information and is used in SecuAPI_SecuBSPToFDx()

```
typedef struct secuapi_finger_data {
    SecuAPI_UINT32      Length;
    SecuAPI_UINT8       FingerID;
    SecuAPI_TEMPLATE_DATA_PTR Template;
} SecuAPI_FINGER_DATA, *SecuAPI_FINGER_DATA_PTR;
```

- **Members**

Length:	Size of structure
FingerID:	Finger position


```
#define SecuAPI_FINGER_ID_UNKNOWN    (0) // for verify
#define SecuAPI_FINGER_ID_RIGHT_THUMB (1)
#define SecuAPI_FINGER_ID_RIGHT_INDEX (2)
#define SecuAPI_FINGER_ID_RIGHT_MIDDLE (3)
#define SecuAPI_FINGER_ID_RIGHT_RING (4)
#define SecuAPI_FINGER_ID_RIGHT_LITTLE (5)
```

```

#define SecuAPI_FINGER_ID_LEFT_THUMB      (6)
#define SecuAPI_FINGER_ID_LEFT_INDEX     (7)
#define SecuAPI_FINGER_ID_LEFT_MIDDLE    (8)
#define SecuAPI_FINGER_ID_LEFT_RING      (9)
#define SecuAPI_FINGER_ID_LEFT_LITTLE    (10)

```

Template: Pointer to minutiae data. See. SecuAPI_TEMPLATE_DATA.

SecuAPI_FIR

Contains SecuGen fingerprint data. The FIR may contain raw sample data, partially processed (intermediate) data, or completely processed data. FIRs 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 encrypted template data is of variable length. TemplateData is the start pointer of the encrypted data.

```

typedef struct secuapi_fir {
    SecuAPI_FIR_FORMAT      Format;
    SecuAPI_FIR_HEADER      Header;
    SecuAPI_FIR_DATA_PTR    Data;
} SecuAPI_FIR, *SecuAPI_FIR_PTR;

```

- **Members**

Format: FIR format. Header and Data structures can have different structure depending on Format. In this version, Format must be set to SecuAPI_FIR_FORMAT_STANDARD.

Header: FIR header. For more information, see SecuAPI_FIR_HEADER structure.

Data: Start pointer for opaque fingerprint data (the data is encrypted). To obtain the data size of fingerprint data, read the data length of the header.

SecuAPI_FIR_DATA

```
typedef SecuAPI_UINT8 SecuAPI_FIR_DATA;
```

SecuAPI_FIR_DATA_TYPE

Mask bits that may be OR'd together to indicate the type of template data in the FIR

```

typedef SecuAPI_UINT16 SecuAPI_FIR_DATA_TYPE;
#define SecuAPI_FIR_DATA_TYPE_RAW      (0x00)
#define SecuAPI_FIR_DATA_TYPE_INTERMEDIATE (0x01)
#define SecuAPI_FIR_DATA_TYPE_PROCESSED (0x02)
#define SecuAPI_FIR_DATA_TYPE_ENCRYPTED (0x10)

```

SecuAPI_FIR_FORMAT

Defines FIR format. In this version, only SecuAPI_FIR_FORMAT_STANDARD is valid. Format may be changed in future releases.

```

typedef SecuAPI_UINT32 SecuAPI_FIR_FORMAT;
#define SecuAPI_FIR_FORMAT_STANDARD      (1) // Used in SecuBSP SDK

#define SecuAPI_FIR_FORMAT_STANDARDPRO   (3) // Used in SecuBSP SDK Pro
#define SecuAPI_FIR_FORMAT_SG400        (4) // Used in SecuBSP SDK Pro

```

SecuAPI_FIR_HEADER

Contains information about FIR

```
typedef struct secuapi_fir_header
{
    SecuAPI_UINT32      Length;           // 4 Bytes
    SecuAPI_UINT32      DataLength;       // 4 Bytes
    SecuAPI_FIR_VERSION Version;          // 2 Bytes
    SecuAPI_FIR_DATA_TYPE DataType;       // 2 Bytes
    SecuAPI_FIR_PURPOSE Purpose;          // 2 Bytes
    SecuAPI_FIR_QUALITY Quality;          // 2 Bytes
    SecuAPI_UINT32      Reserved;         // 4 Bytes
} SecuAPI_FIR_HEADER;
```

- Members**

Length: FIR header length in bytes
 DataLength: FIR data length in bytes
 Version: FIR version. If internal fingerprint data format is changed, this value will be increased.
 Data Type: FIR data type (raw, intermediate, or processed)
 Purpose: Describes the purpose of FIR (e.g. enrollment, verification, or identification)
 Quality: Quality of fingerprint data. Ranges from 1 (lowest) to 100 (highest). (Not used in this release.)
 Reserved: Reserved area for future use

SecuAPI_FIR_HANDLE

A handle to refer to FIR in the SecuBSP

```
typedef SecuAPI_UINT32      SecuAPI_FIR_HANDLE
typedef SecuAPI_FIR_HANDLE* SecuAPI_FIR_HANDLE_PTR
```

SecuAPI_FIR_PAYLOAD

Stores application specific data into FIR

```
typedef struct secuapi_fir_payload {
    SecuAPI_UINT32      Length;
    SecuAPI_UINT8*      Data;
} SecuAPI_FIR_PAYLOAD, *SecuAPI_FIR_PAYLOAD_PTR;
```

- Members**

Length: Length of payload data in bytes
 Data: Start of data buffer pointer

SecuAPI_FIR_PURPOSE

```
typedef SecuAPI_UINT16      SecuAPI_FIR_PURPOSE;
#define SecuAPI_FIR_PURPOSE_VERIFY (1)
#define SecuAPI_FIR_PURPOSE_IDENTIFY (2)
#define SecuAPI_FIR_PURPOSE_ENROLL (3)
#define SecuAPI_FIR_PURPOSE_AUDIT (6)
```


SecuAPI_FIR_QUALITY

```
typedef SecuAPI_UINT16      SecuAPI_FIR_QUALITY;
```

SecuAPI_FIR_FORMAT

```
typedef SecuAPI_UINT32  SecuAPI_FIR_FORMAT;
#define SecuAPI_FIR_FORMAT_STANDARD      (1) // used in SecuBSP SDK

#define SecuAPI_FIR_FORMAT_STANDARDPRO   (3) // Used in SecuBSP SDK Pro
#define SecuAPI_FIR_FORMAT_ANSI378      (4) // Used in SecuBSP SDK Pro
```

MINCONV_DATA_TYPE

```
enum MINCONV_DATA_TYPE
{
    MINCONV_TYPE_SG400,
    MINCONV_TYPE_OLD_FDX
};
```

SecuAPI_FIR_SECURITY_LEVEL

```
typedef SecuAPI_UINT32  SecuAPI_FIR_SECURITY_LEVEL;
#define SecuAPI_FIR_SECURITY_LEVEL_LOWEST      (1)
#define SecuAPI_FIR_SECURITY_LEVEL_LOWER      (2)
#define SecuAPI_FIR_SECURITY_LEVEL_LOW        (3)
#define SecuAPI_FIR_SECURITY_LEVEL_BELOW_NORMAL (4)
#define SecuAPI_FIR_SECURITY_LEVEL_NORMAL      (5)
#define SecuAPI_FIR_SECURITY_LEVEL_ABOVE_NORMAL (6)
#define SecuAPI_FIR_SECURITY_LEVEL_HIGH        (7)
#define SecuAPI_FIR_SECURITY_LEVEL_HIGHER      (8)
#define SecuAPI_FIR_SECURITY_LEVEL_HIGHEST     (9)
```

The Basics of SecuGen Security Levels

Developers should become familiar with the basic concepts behind security levels and their effects on the performance of the fingerprint recognition functions. There will always be a trade-off between user convenience and security.

For example, when you increase the security level, you may experience a higher rate of false rejections (FRR). False rejection occurs when an authorized (registered) user's fingerprint is not successfully matched against the stored sample because more minutiae (fingerprint characteristics) are required for a match. The increased security level raises the matching threshold, which results in a more discriminating fingerprint recognition system.

On the other hand, when you decrease the security level, you may experience a higher rate of false acceptance (FAR). False acceptance is potentially more serious, because this means an unauthorized user could be mistakenly granted access.

The consequence of a "slightly higher security level" could be a slight increase FRR, which is usually nothing more than an inconvenience to users who should re-attempt to match their fingerprints. But the consequence of a "much higher security level" could be a great increase in FRR, which could pose a hindrance to the smooth operation of a fingerprint recognition system. In high security environments where the security level is raised, users will need to be more attentive to physical technique when placing their fingers on the sensor, and they may require basic education regarding environmental considerations such as

bright light and moisture and their possible side effects on fingerprint capturing. In summary, where FRR can be irritating, the net results of FAR can be far more serious if critical resources are at stake.

SecuGen readers use *security levels* to fine-tune the fingerprint matching process, allowing different installations to emphasize security over convenience, or vice versa, depending on the requirements of a site. Security levels range from Lowest (1) to Highest (9). A security setting of “Normal” (5) is considered optimal for many kinds of applications.

Note: FAR and FRR are directly and inversely proportional, so it is important to establish security policies based around this fact.

SecuAPI_FIR_TEXTENCODE

SecuAPI provides another method to use FIR easily with Web or RAD tools. The FIR can be retrieved in text encoded form.

```
typedef struct secuapi_fir_textencode {
    SecuAPI_BOOL        IsWideChar;
    SecuAPI_CHAR_PTR    TextFIR;
} SecuAPI_FIR_TEXTENCODE, *SecuAPI_FIR_TEXTENCODE_PTR;
```

- Members**

IsWideChar: Indicates whether or not TextFIR is wide char. If TextFIR form is wide char form, it has SecuAPI_TRUE, otherwise SecuAPI_FALSE.

TextFIR: Text encoded FIR

SecuAPI_FIR_VERSION

```
typedef SecuAPI_UINT16 SecuAPI_FIR_VERSION;
```

SecuAPI_HANDLE

A handle to refer to SecuBSP module

```
typedef SecuAPI_UINT32    SecuAPI_HANDLE
typedef SecuAPI_HANDLE *  SecuAPI_HANDLE_PTR
```

```
#define    SecuAPI_INVALID_HANDLE    (0)
```

SecuAPI_HWND

```
#ifdef WIN32
typedef HWND              SecuAPI_HWND;
#else
typedef SecuAPI_UINT32    SecuAPI_HWND;
#endif
```

SecuAPI_IMAGE_DATA

Contains image data of a fingerprint and is used in SecuAPI_EXPORT_AUDIT_DATA

```
typedef struct secuapi_image_data {
```

```

        SecuAPI_UINT32      Length;
        SecuAPI_UINT8*      Data;
    } SecuAPI_IMAGE_DATA, *SecuAPI_IMAGE_DATA_PTR;

```

- **Members**

Length: Size of structure

Data: Contains image data for a finger. Total image data size can vary and depends on number of samples per finger, image width and image height. For example, If image width and height are 260 and 300, samples per finger is 2, total image size for a finger will be *2*260* 300 bytes.

SecuAPI_INIT_INFO_0

Globally initializes each variable in the SecuGen module

```

typedef struct secuapi_init_info_0 {
    SecuAPI_UINT32      StructureType;
    SecuAPI_UINT32      MaxFingersForEnroll;
    SecuAPI_UINT32      SamplesPerFinger;
    SecuAPI_UINT32      DefaultTimeout;
    SecuAPI_UINT32      EnrollImageQuality;
    SecuAPI_UINT32      VerifyImageQuality;
    SecuAPI_UINT32      IdentifyImageQuality;
    SecuAPI_FIR_SECURITY_LEVEL SecurityLevel;
} SecuAPI_INIT_INFO_0, *SecuAPI_INIT_INFO_PTR;

```

- **Members**

StructureType: Must be 0

MaxFingersForEnroll: Maximum fingers for enrollment. Value can be 1 to 10. Default is 10.

SamplesPerFinger: Samples per finger. Default is 2. For verification time, 1 sample is captured.

DefaultTimeout: Default timeout value in milliseconds, used for capturing fingerprint image data from device. Default is 10000 (10sec).

EnrollImageQuality: FIR quality on enrollment time. Ranges from 30 (lowest) to 100 (highest). Default is 50.

VerifyImageQuality: FIR quality on verification time. Ranges from 0 (lowest) to 100 (highest). Default is 30.

IdentifyImageQuality: FIR quality on identification time. Ranges from 0 (lowest) to 100 (highest). Default is 50.

SecurityLevel: Security level for verification. Ranges from 1 (Lowest) to 10 (Highest). Default is 5 (Normal). As the value is increased, the FRR increases while FAR decreases. For more information, see SecuAPI_FIR_SECURITY_LEVEL.

SecuAPI_INIT_INFO_PTR

```

typedef SecuAPI_VOID_PTR SecuAPI_INIT_INFO_PTR;

```

SecuAPI_INPUT_FIR

A structure used to input a FIR to the API. Such input can be in one of three forms: a FIR handle (SecuAPI_FIR_FORM_HANDLE), an actual FIR (SecuAPI_FIR_FORM_FULLFIR), or an actual FIR with encoded text (FIR_FORM_TEXT_ENCODE). The form of FIR is defined in SecuAPI_INPUT_FIR_FORM.

```

typedef struct secuapi_input_fir {
    SecuAPI_INPUT_FIR_FORM      FIRForm,
    Union {
        SecuAPI_FIR_HANDLE_PTR      FIRinBSP;
    }
}

```

```

        SecuAPI_VOID_PTR          FIR,
        SecuAPI_FIR_TEXTENCOD_PTR TextFIR;
    }
} SecuAPI_INPUT_FIR, *SecuAPI_INPUT_FIR_PTR;

```

SecuAPI_INPUT_FIR_FORM

```

typedef SecuAPI_UINT8 SecuAPI_INPUT_FIR_FORM
#define SecuAPI_FIR_FORM_HANDLE          (2)
#define SecuAPI_FIR_FORM_FULLFIR        (3)
#define SecuAPI_FIR_FORM_TEXTENCOD      (4)

```

SecuAPI_MAX_DEVICE

```

#define SecuAPI_MAX_DEVICE                (0xfe)

```

SecuAPI_RETURN

```

typedef SecuAPI_UINT32 SecuAPI_RETURN;

```

SecuAPI_TEMPLATE_DATA

Contains minutiae data of a fingerprint and is used in SecuAPI_SecuBSPToFDx()

```

typedef struct secuapi_image_data {
    SecuAPI_UINT32      Length;
    SecuAPI_UINT8*      Data;
} SecuAPI_IMAGE_DATA, *SecuAPI_IMAGE_DATA_PTR;

```

- **Members**

Length: Size of structure

Data: Pointer to minutiae data. The data size can vary and is based on number of sample per finger. The size of one sample is 400 bytes. For example, if samples per finger is 2, total data size will be 2*400 bytes.

SecuAPI_VERSION

```

typedef struct secuapi_version {
    SecuAPI_UINT32      Major;
    SecuAPI_UINT32      Minor;
} SecuAPI_VERSION;

```

SecuAPI_WINDOW_CALLBACK

```

typedef SecuAPI_RETURN (WINAPI* SecuAPI_WINDOW_CALLBACK)
(SecuAPI_WINDOW_CALLBACK_PARAM_PTR, SecuAPI_VOID_PTR);

```

SecuAPI_WINDOW_CALLBACK_INFO

Sets callback function information. Refer to SecuAPI_WINDOW_OPTION structure.

```

typedef struct secuapi_callback_info {
    SecuAPI_UINT32      CallBackType;
    SecuAPI_WINDOW_CALLBACK CallBackFunction;
}

```

```

        SecuAPI_VOID_PTR        UserCallBackParam;
    } SecuAPI_CALLBACK_INFO, *SecuAPI_CALLBACK_INFO_PTR;

```

- **Members**

CallBackType: Type of callback function. If value set to 0, the first parameter is assigned to SecuAPI_WINDOW_CALLBACK_PARAM_0. If value set to 1, the parameter is assigned to SecuAPI_WINDOW_CALLBACK_PARAM_1.

CallBackFunction: Function name

UserCallBackParam: User information for callback function. The second parameter is the pointer of user information.

SecuAPI_WINDOW_CALLBACK_PARAM_0

```

typedef struct secuapi_window_callback_param_0 {
    SecuAPI_UINT32        dwQuality;
    SecuAPI_UINT8*        lpImageBuf;
    SecuAPI_VOID_PTR      lpReserved;
} SecuAPI_WINDOW_CALLBACK_PARAM_0, *SecuAPI_WINDOW_CALLBACK_PARAM_PTR_0;

```

SecuAPI_WINDOW_CALLBACK_PARAM_1

```

typedef struct secuapi_window_callback_param_1 {
    SecuAPI_UINT32        Result;
    SecuAPI_VOID_PTR      Reserved;
} SecuAPI_WINDOW_CALLBACK_PARAM_1, *SecuAPI_WINDOW_CALLBACK_PARAM_PTR_1;

```

SecuAPI_WINDOW_CALLBACK_PARAM_PTR

```

typedef SecuAPI_VOID_PTR SecuAPI_WINDOW_CALLBACK_PARAM_PTR;

```

SecuAPI_WINDOW_STYLE

Changes SecuBSP's window style while capturing fingerprint image

```

typedef SecuAPI_UINT32        SecuAPI_WINDOW_STYLE
#define SecuAPI_WINDOW_STYLE_POPUP                (0)
#define SecuAPI_WINDOW_STYLE_INVISIBLE            (1)
#define SecuAPI_WINDOW_STYLE_CONTINUOUS           (2)

/* OR flag used only in high 2 bytes. */
#define SecuAPI_WINDOW_STYLE_NO_FPIMG              (0x00010000)
#define SecuAPI_WINDOW_STYLE_TOPMOST              (0x00020000)
#define SecuAPI_WINDOW_STYLE_NO_WELCOME           (0x00040000)

```

SecuAPI_WINDOW_STYLE_NO_WELCOME is used only for Enroll()

SecuAPI_WINDOW_OPTION

Allows close control of the BSP user interface: the wizards for the AdjustDevice(), Capture(), Enroll() and Verify() functions.

```

typedef struct secuapi_window_option {
    SecuAPI_UINT32        Length;
    SecuAPI_WINDOW_STYLE  WindowStyle;
}

```

```

        SecuAPI_HWND                ParentWnd;
        SecuAPI_HWND                FingerWnd;
        SecuAPI_CALLBACK_INFO       CaptureCallBackInfo;
        SecuAPI_CALLBACK_INFO       FinishCallBackInfo;
        SecuAPI_CHAR_PTR             CaptionMsg;
        SecuAPI_CHAR_PTR             CancelMsg;
        SecuAPI_VOID_PTR             Reserved;
    } SecuAPI_WINDOW_OPTION, *SecuAPI_WINDOW_OPTION_PTR;

```

- **Members**

Length: Length of structure

WindowStyle: Style of dialog window

SecuAPI_WINDOW_STYLE_POPUP
Pop-up window style (default style)

SecuAPI_WINDOW_STYLE_INVISIBLE
Used to show fingerprint image in window provided by application. If this value is used, fingerprint image will be displayed in FingerWnd (4th parameter) provided from application instead of window provided from SecuBSP. This value can only be used in the SecuAPI_Capture() function.

SecuAPI_WINDOW_STYLE_CONTINUOUS
Pop-up window style, but parent window disappears during image capture. The pop-up window will occupy the same position as the parent window. Continuity is maintained if the sizes of the parent and BSP windows are the same.

ParentWnd: Parent window's handle

FingerWnd: The handle of the fingerprint image window. This variable is used only in SecuAPI_WINDOW_STYLE_INVISIBLE and SecuAPI_Capture() functions.

CaptureCallBackInfo: Information of the callback function. Callback function will be called whenever a frame is captured. CallBackType must be set to 0 for CaptureCallBackInfo. This variable is used only in SecuAPI_WINDOW_STYLE_INVISIBLE and SecuAPI_Capture() functions.

FinishCallBackInfo: Information related to the callback function. Callback function will be called just before windows are closed. CallBackType must be set to 1 for CaptureCallBackInfo.

CaptionMsg: Caption message of information dialog

CancelMsg: Cancel message when the "Cancel" button is clicked

Reserved: Reserved Area

2.3. Error Constants

SecuAPI has three types of errors: general, device, and user interface errors. Each of these error categories has a different base value. Error codes are derived by adding the error number to its corresponding base value. For example, SecuAPIERROR_INVALID_HANDLE is error 0x01, and so 0x01 would be added to the SecuAPIERROR_BASE_GENERAL value, 0x0000, to derive the hexadecimal value of 0x0001.

Error types defined (General, Device, UI)	Base value
#define SecuAPIERROR_BASE_GENERAL	0x0000
#define SecuAPIERROR_BASE_DEVICE	0x0100
#define SecuAPIERROR_BASE_UI	0x0200

GENERAL ERRORS		
Error Code	Error name	Description
0x0000	SecuAPIERROR_NONE	Function completed successfully
0x0001	SecuAPIERROR_INVALID_HANDLE	Invalid handle in input function parameter or input field of a data structure
0x0002	SecuAPIERROR_INVALID_POINTER	Invalid pointer in input function parameter or input field of a data structure
0x0003	SecuAPIERROR_INVALID_TYPE	Input structure type is invalid
0x0004	SecuAPIERROR_FUNCTION_FAIL	Function failed for unknown reason (internal)
0x0005	SecuAPIERROR_STRUCTTYPE_NOT_MATCHED	Type of input structure does not match request type value
0x0006	SecuAPIERROR_ALREADY_PROCESSED	Template was already processed
0x0007	SecuAPIERROR_EXTRACTION_OPEN_FAIL	Extraction module cannot be opened
0x0008	SecuAPIERROR_VERIFICATION_OPEN_FAIL	Verification module cannot be opened
0x0009	SecuAPIERROR_DATA_PROCESS_FAIL	Internal error occurred during data processing
0x000a	SecuAPIERROR_MUST_BE_PROCESSED_DATA	Function requires a fully processed FIR
0x000b	SecuAPIERROR_INTERNAL_CHECKSUM_FAIL	Checksum of FIR data in input parameter is invalid
0x000c	SecuAPIERROR_ENCRYPTED_DATA_ERROR	Encrypted FIR data in input parameter is broken
0x000e	SecuAPIERROR_UNKNOWN_VERSION	Input parameter contains an unknown version of FIR data
0x000f	SecuAPIERROR_VALIDITY_FAIL	Not currently used
0x0010	SecuAPIERROR_INIT_MAXFINGER	Maximum finger count for enrollment is invalid. Must be set in range of 1~10.
0x0011	SecuAPIERROR_INIT_SAMPLESPERFINGER	Samples-per-finger count for enrollment is invalid. Must be set in range of 1~10.
0x0012	SecuAPIERROR_INIT_ENROLLQUALITY	Image quality value for enrollment is invalid. Must be set in range of 0~100.
0x0013	SecuAPIERROR_INIT_VERIFYQUALITY	Image quality value for verification is invalid. Must be set in range of 1~100.
0x0014	SecuAPIERROR_INIT_IDENTIFYQUALITY	Image quality value for identification is invalid. Must be set in range of 0~100.
0x0015	SecuAPIERROR_INIT_SECURITYLEVEL	Security level value is invalid. Use range from SecuAPI_FIR_SECURITY_LEVEL_LOWEST(1) to SecuAPI_FIR_SECURITY_LEVEL_HIGHEST(9) to set security level value

Device Errors		
Error Code	Error name	Description
0x0101	SecuAPIERROR_DEVICE_OPEN_FAIL	Device could not be opened
0x0102	SecuAPIERROR_INVALID_DEVICE_ID	Device ID in input parameter is invalid
0x0103	SecuAPIERROR_WRONG_DEVICE_ID	Device ID in input parameter refers to a different device
0x0104	SecuAPIERROR_DEVICE_ALREADY_OPENED	Device in input parameter is already opened
0x0105	SecuAPIERROR_DEVICE_NOT_OPENED	Device in input parameter is not yet opened
0x0106	SecuAPIERROR_DEVICE_BRIGHTNESS	Brightness value is invalid. Must be set in range of 0~100.
0x0107	SecuAPIERROR_DEVICE_CONTRAST	Contrast value is invalid. Must be set in range of 0~100.
0x0108	SecuAPIERROR_DEVICE_GAIN	Gain value is invalid. Must be set to 1, 2, 4, or 8.

USER INTERFACE ERRORS		
Error Code	Name	Description
0x0201	SecuAPIERROR_USER_CANCEL	Use of "Cancel" button closed function
0x0202	SecuAPIERROR_USER_BACK	Use of "Back" button closed function
0x0203	SecuAPIERROR_CAPTURE_TIMEOUT	Fingerprint capture process timed out, closing the function

CHAPTER 3. Functions

3.1. Basic Functions

SecuAPI_Init

```
SecuAPI_RETURN SecuAPI_Init(
    OUT   SecuAPI_HANDLE_PTR   Handle);
```

- **Description**

Initializes the SecuBSP module and returns the module handle. This function must be called once before any other function can be called.

- **Parameters**

Handle: The SecuBSP module handle

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_POINTER

There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPI_Terminate

```
SecuAPI_RETURN SecuAPI_Terminate(
    IN   SecuAPI_HANDLE   Handle);
```

- **Description**

Terminates the caller's use of the BSP module. The module can then clean up all internal states associated with the calling application. This function must be called once for each call to SecuAPI_Init().

- **Parameters**

Handle: The handle of the SecuBSP module

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure

SecuAPI_GetVersion

```
SecuAPI_RETURN SecuAPI_GetVersion(
    IN   SecuAPI_HANDLE   Handle,
    OUT  SecuAPI_VERSION_PTR   Version
);
```

- **Description**

Returns current BSP module version

- **Parameters**

Handle: The handle of the BSP module

Version: The buffer pointer containing version information

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure

SecuAPIERROR_INVALID_POINTER

There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPI_GetInitInfo

```
SecuAPI_RETURN SecuAPI_GetInitInfo(
    IN     SecuAPI_HANDLE   Handle,
    IN     SecuAPI_UINT8    StructureType,
    OUT    SecuAPI_INIT_INFO_PTR InitInfo
);
```

- **Description**

Gets initialization information

- **Parameters**

Handle: The handle of the SecuBSP module

StructureType: The structure type selector. Must be 0.

InitInfo: The structure pointer for init info

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure

SecuAPIERROR_INVALID_POINTER

There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPIERROR_STRUCTTYPE_NOT_MATCHED The type of input structure does not match the request type value. It occurs when StructureType and StructureType in InitInfo are different

SecuAPIERROR_INVALID_TYPE

The input structure type is invalid (structure type is not a zero)

SecuAPI_SetInitInfo

```
SecuAPI_RETURN SecuAPI_SetInitInfo(
    IN     SecuAPI_HANDLE   Handle,
    IN     SecuAPI_UINT8    StructureType,
    OUT    SecuAPI_INIT_INFO_PTR InitInfo);
```

- **Description**

Sets InitInfo structure. In this version, only Structuretype 0 is supported. Before calling this function, the InitInfo structure must be initialized.

- **Parameters**

Handle: The handle of SecuBSP module

StructureType: Structure type. Must be 0.

InitInfo: The pointer of InitInfo structure

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
<i>SecuAPIERROR_STRUCTTYPE_NOT_MATCHED</i>	The type of input structure does not match the request type value. It occurs when <i>StructureType</i> and <i>StructureType</i> in <i>InitInfo</i> are different.
<i>SecuAPIERROR_INVALID_TYPE</i>	The input structure type is invalid (structure type is not a zero)
<i>SecuAPIERROR_INIT_MAXFINGER</i>	The maximum finger count for enrollment is invalid. Must be set in the range of 1~10.
<i>SecuAPIERROR_INIT_SAMPLESPERFINGER</i>	The samples-per-finger count for enrollment is invalid. Must be set in the range of 1~10.
<i>SecuAPIERROR_INIT_ENROLLQUALITY</i>	The image quality value for enrollment is invalid. Must be set in the range of 30~100.
<i>SecuAPIERROR_INIT_VERIFYQUALITY</i>	The image quality value for verification is invalid. Must be set in the range of 1~100.
<i>SecuAPIERROR_INIT_IDENTIFYQUALITY</i>	The image quality value for identification is invalid. Must be set in the range of 0~100.
<i>SecuAPIERROR_INIT_SECURITYLEVEL</i>	The security level value is invalid. Use the range from <i>SecuAPI_FIR_SECURITY_LEVEL_LOWEST</i> (1) to <i>SecuAPI_FIR_SECURITY_LEVEL_HIGHEST</i> (9) to set the security level value.

SecuAPI_EnumerateDevice

```

SecuAPI_RETURN SecuAPI_EnumerateDevice (
    IN    SecuAPI_HANDLE      Handle,
    OUT   SecuAPI_UINT32*     NumDevice,
    OUT   SecuAPI_DEVICE_ID** DeviceID
);

```

- **Description**

Gets the number and ID of readers attached to the system

- **Parameters**

Handle: The handle of the SecuBSP module

NumDevice: Number of attached readers

DeviceID: Pointer to storage buffer of Device ID list. The buffer for the Device ID list is allocated and managed by BSP. Developers are not responsible for releasing Device ID list memory, since this memory is automatically cleared by calling the *SecuAPI_Terminate()* function.

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPI_GetDeviceInfo

```

SecuAPI_RETURN SecuAPI_GetDeviceInfo(
    IN    SecuAPI_HANDLE      Handle,
    IN    SecuAPI_DEVICE_ID   DeviceID,

```

```

        IN    SecuAPI_UINT8          StructureType,
        OUT   SecuAPI_DEVICE_INFO_PTR DeviceInfo
    );

```

- **Description**

Gets information for a specified reader

- **Parameters**

Handle: The handle of the SecuBSP module

DeviceID: Device ID to get information from

StructureType: Structure type; must be 0. (SecuAPI_DEVICE_INFO_0)

DeviceInfo: The structure pointer of device info

- **Returned Values**

SecuAPIERROR_NONE The function was successful

SecuAPIERROR_INVALID_HANDLE There is an invalid handle in an input function parameter or input field of a data structure

SecuAPIERROR_INVALID_POINTER There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPIERROR_DEVICE_NOT_OPENED The device in an input parameter is not yet opened

SecuAPIERROR_WRONG_DEVICE_ID The device ID in an input parameter refers to a different device

SecuAPIERROR_STRUCTTYPE_NOT_MATCHED The type of input structure does not match the request type value. It occurs when *StructureType* and *StructureType* in *InitInfo* are different.

SecuAPIERROR_INVALID_TYPE The input structure type is invalid (structure type is not a zero)

SecuAPI_SetDeviceInfo

```

SecuAPI_RETURN SecuAPI_SetDeviceInfo(
    IN    SecuAPI_HANDLE          Handle,
    IN    SecuAPI_DEVICE_ID       DeviceID,
    IN    SecuAPI_UINT8          StructureType,
    IN    SecuAPI_DEVICE_INFO_PTR DeviceInfo
);

```

- **Description**

Sets specific information for current attached device. Image width and height cannot be set.

- **Parameters**

Handle: The handle of the SecuBSP module

DeviceID: Device ID to get information from

StructureType: Structure type; must be 0 (SecuAPI_DEVICE_INFO_0)

DeviceInfo: The structure pointer for device info

- **Returned Values**

SecuAPIERROR_NONE The function was successful

SecuAPIERROR_INVALID_HANDLE There is an invalid handle in an input function parameter or input field of a data structure

SecuAPIERROR_INVALID_POINTER There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPIERROR_DEVICE_NOT_OPENED The device in an input parameter is not yet opened

SecuAPIERROR_WRONG_DEVICE_ID The device ID in an input parameter refers to a different device

SecuAPIERROR_STRUCTTYPE_NOT_MATCHED The type of input structure does not match the request type value. It occurs when *StructureType* and *StructureType* in *InitInfo* are different.

<i>SecuAPIERROR_INVALID_TYPE</i>	The input structure type is invalid (structure type is not a zero)
<i>SecuAPIERROR_DEVICE_BRIGHTNESS</i>	The brightness value is invalid. Must be set in the range of 0~100.
<i>SecuAPIERROR_DEVICE_CONTRAST</i>	The contrast value is invalid. Must be set in the range of 0~100.
<i>SecuAPIERROR_DEVICE_GAIN</i>	The gain value is invalid. Must be set to 1, 2, 4, or 8.

SecuAPI_OpenDevice

```
SecuAPI_RETURN SecuAPI_OpenDevice(
    IN      SecuAPI_HANDLE    Handle,
    IN      SecuAPI_DEVICE_ID DeviceID
);
```

- **Description**

Initializes specified reader for SecuBSP to use. If Device ID is 0, default reader is used. If this function is not called, SecuBSP uses default reader.

- **Parameters**

Handle: The handle of the SecuBSP module

DeviceID: Device to open

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_DEVICE_ID</i>	The device ID in an input parameter is invalid
<i>SecuAPIERROR_DEVICE_ALREADY_OPENED</i>	The device in an input parameter is already opened
<i>SecuAPIERROR_DEVICE_OPEN_FAIL</i>	Device could not be opened

SecuAPI_CloseDevice

```
SecuAPI_RETURN SecuAPI_CloseDevice(
    IN      SecuAPI_HANDLE    Handle,
    IN      SecuAPI_DEVICE_ID DeviceID
);
```

- **Description**

Detaches specified reader

- **Parameters**

Handle: The handle of the SecuBSP module

DeviceID: Device to close

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_DEVICE_NOT_OPENED</i>	The device in an input parameter is not yet opened
<i>SecuAPIERROR_WRONG_DEVICE_ID</i>	The device ID in an input parameter is not a currently opened device

SecuAPI_AdjustDevice

```
SecuAPI_RETURN SecuAPI_AdjustDevice(
```

```

        IN      SecuAPI_HANDLE          Handle,
        IN      const SecuAPI_WINDOW_OPTION_PTR  WindowOption
    );

```

- **Description**

Adjusts SecuGen fingerprint reader to capture optimum quality image

- **Parameters**

Handle: The handle of the SecuBSP module

WindowOption: Refer to SecuAPI_WINDOW_OPTION structure

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_DEVICE_NOT_OPENED</i>	The device in an input parameter is not yet opened
<i>SecuAPIERROR_WRONG_DEVICE_ID</i>	The device ID in an input parameter refers to a different device
<i>SecuAPIERROR_USER_CANCEL</i>	Use of "Cancel" button closed function

SecuAPI_MonitorDevice

```

SecuAPI_RETURN SecuAPI_MonitorDevice(
    IN      SecuAPI_HANDLE          Handle,
    IN      SecuAPI_DEVICE_ID       DeviceID
    IN      SecuAPI_BOOL            Enable,
    IN      SecuAPI_HWND            hWnd
);

```

- **Description**

Enables/disables monitoring, for events such as Auto-On. This function is not supported by FDP0x-, FDU01- or FDU02-based readers.

- **Parameters**

Handle: The handle of the SecuBSP module

DeviceID: Device to monitor

Enable: SecuAPI_TRUE to enable device monitoring, otherwise SecuAPI_FALSE

hWnd: Window handle to receive message from the device

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_DEVICE_NOT_OPENED</i>	The device in an input parameter is not yet opened
<i>SecuAPIERROR_WRONG_DEVICE_ID</i>	The device ID in an input parameter refers to a different device

3.2. Memory Functions

SecuAPI_FreeFIRHandle

```

SecuAPI_RETURN SecuAPI_FreeFIRHandle (
    IN      SecuAPI_HANDLE          Handle,
    IN      SecuAPI_FIR_HANDLE      FIRHandle
);

```

- **Description**
Frees memory allocated to FIRHandle in SecuBSP module. After calling function to get FIR, this function must be called to free the memory.
- **Parameters**
Handle: The handle of SecuBSP module
FIRHandle: FIR handle in SecuBSP module
- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure (Handle or FIRHandle)

SecuAPI_GetFIRFromHandle

```
SecuAPI_RETURN SecuAPI_GetFIRFromHandle (
    IN          SecuAPI_HANDLE      Handle,
    IN          SecuAPI_FIR_HANDLE  FIRHandle);
    OUT         SecuAPI_FIR_PTR     FIR
);
```

- **Description**
Gets FIR from the FIRHandle in SecuBSP module. Application must assign a buffer for the SecuAPI_FIR structure. In this version, this function is exactly the same as SecuAPI_GetExtendedFIRFromHandle except for the Format parameter. **Note**: It is recommended to use SecuAPI_GetExtendedFIRFromHandle() to maintain compatibility with future releases.

For more information, see SecuAPI_GetExtendedFIRFromHandle().

- **Parameters**
Handle: The handle of the SecuBSP module
FIRHandle: FIR handle in the SecuBSP module
FIR: FIR buffer assigned by application
- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure (Handle or FIRHandle)
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPI_GetHeaderFromHandle

```
SecuAPI_RETURN SecuAPI_GetHeaderFromHandle (
    IN          SecuAPI_HANDLE      Handle,
    IN          SecuAPI_FIR_HANDLE  FIRHandle);
    OUT         SecuAPI_FIR_HEADER_PTR Header
);
```

- **Description**
Retrieves FIR header information from the FIRHandle in the SecuBSP module. Application must assign buffer for SecuAPI_FIR_HEADER. In this version, this function is exactly the same as SecuAPI_GetExtendedHeaderFromHandle except for the Format parameter. **Note**: It is recommended to

use SecuAPI_GetExtendedHeaderFromHandle() to maintain compatibility with future releases.

For more information, see SecuAPI_GetExtendedHeaderFromHandle().

- **Parameters**

Handle: The handle of the SecuBSP module

FIRHandle: FIR handle in the SecuBSP module

Header: FIR header buffer provided by the application

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure (Handle or FIRHandle)

SecuAPIERROR_INVALID_POINTER

There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPI_GetExtendedFIRFromHandle

```
SecuAPI_RETURN SecuAPI_GetExtendedFIRFromHandle (
    IN          SecuAPI_HANDLE      Handle,
    IN          SecuAPI_FIR_HANDLE  FIRHandle);
    OUT         SecuAPI_VOID_PTR    FIR,
    IN          SecuAPI_FIR_FORMAT  Format
);
```

- **Description**

Gets FIR specified in Format parameter from the FIRHandle in SecuBSP module. Application must assign the buffer of SecuAPI_FIR structure.

- **Parameters**

Handle: The handle of the SecuBSP module

FIRHandle: FIR handle in the SecuBSP module

FIR: FIR buffer assigned by application

Format: FIR format.

In SecuBSP SDK, format must be SecuAPI_FIR_FORMAT_STANDARD.

In SecuBSP SDK *Pro*, format may be either SecuAPI_FIR_FORMAT_STANDARDPRO or SecuAPI_FIR_FORMAT_ANSI378. The default format is SecuAPI_FIR_FORMAT_STANDARDPRO so, if SecuAPI_GetFIRFromHandle is used, the FIR format will be SecuAPI_FIR_FORMAT_STANDARDPRO.

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure (Handle or FIRHandle)

SecuAPIERROR_INVALID_POINTER

There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPI_GetExtendedHeaderFromHandle

```
SecuAPI_RETURN SecuAPI_GetExtendedHeaderFromHandle (
    IN          SecuAPI_HANDLE      Handle,
    IN          SecuAPI_FIR_HANDLE  FIRHandle,
    OUT         SecuAPI_VOID_PTR    Header,
```



```

        IN          SecuAPI_FIR_FORMAT    Format
    );

```

- **Description**

Gets FIR header specified in Format parameter from the FIRHandle in SecuBSP module. Application must assign buffer for SecuAPI_FIR_HEADER.

- **Parameters**

Handle: The handle of the SecuBSP module

FIRHandle: FIR handle in the SecuBSP module

Header: FIR header buffer provided by the application

Format: FIR format.

In SecuBSP SDK, format must be SecuAPI_FIR_FORMAT_STANDARD.

In SecuBSP SDK *Pro*, format may be either SecuAPI_FIR_FORMAT_STANDARDPRO or SecuAPI_FIR_FORMAT_ANSI378. The default format is SecuAPI_FIR_FORMAT_STANDARDPRO so, if SecuAPI_GetHeaderFromHandle is used, the header format will be SecuAPI_FIR_FORMAT_STANDARDPRO.

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure (Handle or FIRHandle)

SecuAPIERROR_INVALID_POINTER

There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPI_FreeFIR

```

SecuAPI_RETURN SecuAPI_FreeFIR (
    IN          SecuAPI_HANDLE    Handle,
    IN          SecuAPI_VOID_PTR  FIR
);

```

- **Description**

Frees memory allocated to the FIR; is used after calling GetFIRFromHandle()

- **Parameters**

Handle: The handle of the SecuBSP module

FIR: Pointer of FIR

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure

SecuAPI_GetTextFIRFromHandle

```

SecuAPI_RETURN SecuAPI_GetTextFIRFromHandle (
    IN          SecuAPI_HANDLE    Handle,
    IN          SecuAPI_FIR_HANDLE FIRHandle,
    OUT         SecuAPI_FIR_TEXTENCODING_PTR TextFIR,
    IN          SecuAPI_BOOL      IsWideChar
);

```

- **Description**

Gets text encoded FIR data from the FIRHandle in SecuBSP module. Application must assign buffer for TextFIR. In this version, this function is exactly the same as SecuAPI_GetExtendedTextFIRFromHandle except Format parameter. **Note:** It is recommended to use SecuAPI_GetExtendedTextFIRFromHandle() to maintain compatibility with future releases.

For more information, see SecuAPI_GetExtendedTextFIRFromHandle().

- **Parameters**

Handle: The handle of the SecuBSP module

FIRHandle: FIR handle in SecuBSP module

TextFIR: The buffer pointer containing text encoded FIR

IsWideChar: Specifies if application uses Unicode characters or not. If it uses Unicode, then it specifies SecuAPI_TRUE, else SecuAPI_FALSE.

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure (Handle or FIRHandle)

SecuAPIERROR_INVALID_POINTER

There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPI_GetExtendedTextFIRFromHandle

```
SecuAPI_RETURN SecuAPI_GetExtendedTextFIRFromHandle (
    IN          SecuAPI_HANDLE      Handle,
    IN          SecuAPI_FIR_HANDLE  FIRHandle,
    OUT         SecuAPI_FIR_TEXTENCODING_PTR  TextFIR,
    IN          SecuAPI_BOOL        IsWideChar,
    IN          SecuAPI_FIR_FORMAT  Format
);
```

- **Description**

Retrieves the FIR in a text encoded format rather than the standard C structure. The application allocates memory for a string buffer and then passes this as an argument, along with the FIR handle. The BSP returns the fingerprint data in encoded text format.

- **Parameters**

Handle: The handle of the SecuBSP module

FIRHandle: FIR handle in SecuBSP module

TextFIR: The buffer pointer containing text encoded FIR

IsWideChar: Specifies if application uses Unicode characters or not. If it uses Unicode, then it specifies SecuAPI_TRUE, else SecuAPI_FALSE.

Format: FIR format.

In SecuBSP SDK, format must be SecuAPI_FIR_FORMAT_STANDARD.

In SecuBSP SDK *Pro*, format may be either SecuAPI_FIR_FORMAT_STANDARDPRO or SecuAPI_FIR_FORMAT_ANSI378. The default format is SecuAPI_FIR_FORMAT_STANDARDPRO so, if SecuAPI_GetTextFIRFromHandle is used, the FIR data format will be SecuAPI_FIR_FORMAT_STANDARDPRO.

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_HANDLE

There is an invalid handle in an input function parameter or input field of a data structure (Handle or FIRHandle)

<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
-------------------------------------	---

SecuAPI_FreeTextFIR

```
SecuAPI_RETURN SecuAPI_FreeTextFIR (
    IN          SecuAPI_HANDLE          Handle,
    IN          SecuAPI_FIR_TEXTENCODING_PTR  TextFIR
);
```

- **Description**

Frees memory allocated to text encoded FIR; is used after calling GetTextFIRFromHandle()

- **Parameters**

Handle: The handle of the SecuBSP module
TextFIR: Pointer of text encoded FIR

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure

SecuAPI_FreePayload

```
SecuAPI_RETURN SecuAPI_FreeFIR (
    IN          SecuAPI_HANDLE          Handle,
    IN          SecuAPI_FIR_PAYLOAD_PTR  Payload
);
```

- **Description**

Frees memory allocated to FIR payload

- **Parameters**

Handle: The handle of the SecuBSP module
Payload: Payload buffer pointer

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure

SecuAPI_FreeExportData

```
SecuAPI_RETURN SecuAPI_FreeExportData (
    IN          SecuAPI_HANDLE          Handle,
    IN          SecuAPI_EXPORT_DATA_PTR pExportData
);
```

- **Description**

Frees memory allocated to exported data; is used after calling SecuAPI_SecuBSPToFDx()

- **Parameters**

Handle: The handle of the SecuBSP module
pExportData: The pointer of SecuAPI_EXPORT_DATA structure

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure

SecuAPI_FreeExportAuditData

```
SecuAPI_RETURN SecuAPI_FreeExportAuditData (
    IN          SecuAPI_HANDLE          Handle,
    IN          SecuAPI_EXPORT_AUDIT_DATA_PTR  pExportAuditData
);
```

- **Description**

Frees memory allocated to exported data; is used after calling *SecuAPI_SecuBSPTolmage* ()

- **Parameters**

Handle: The handle of the SecuBSP module

pExportAuditData: The pointer of *SecuAPI_EXPORT_AUDIT_DATA* structure

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure

3.3. BSP Functions

SecuAPI_Capture

```
SecuAPI_RETURN SecuAPI_Capture(
    IN          SecuAPI_HANDLE          Handle,
    IN          SecuAPI_FIR_PURPOSE     Purpose,
    OUT         SecuAPI_FIR_HANDLE_PTR  CapturedFIR,
    IN          SecuAPI_SINT32          Timeout,
    OUT         SecuAPI_FIR_HANDLE_PTR  AuditData,
    IN          const SecuAPI_WINDOW_OPTION_PTR  WindowOption);
```

- **Description**

Captures samples for the purpose specified and can return different types of FIR. The Purpose is recorded in the header of the CapturedFIR (e.g., enrollment, verification, identification). If AuditData is non-NULL, a FIR of type “raw” may be returned. The function returns handles to whatever data is collected, and all local operations can be completed using the handles.

- **Parameters**

Handle: The handle of the SecuBSP module

Purpose: The Purpose of the capture. The UI displayed to users depends on this value, and is different for enrollment, verification, and identification.

- *SecuAPI_FIR_PURPOSE_VERIFY*: Data captured for verification
- *SecuAPI_FIR_PURPOSE_IDENTIFY*: Data captured for identification
- *SecuAPI_FIR_PURPOSE_ENROLL*: Data captured for enrollment
- *SecuAPI_FIR_PURPOSE_AUDIT*: Data captured for audit. Only raw image data will be stored in FIR.

CapturedFIR: A handle to a FIR containing captured data. This data is either an “intermediate” type FIR (which can be used only by the *Process* or *CreateTemplate*() functions, depending on the purpose) or a “processed” FIR (which can be used directly by *VerifyMatch*, depending on the purpose). **Note**: in BSP

version 1.0, the capture function returns a processed FIR.

Timeout: 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. If value is -1, then default value will be used.

AuditData: A handle to a FIR containing raw fingerprint image data. This data may be used to provide human-identifiable data of the person at the reader. If the pointer is NULL on input, no audit data is collected.

WindowOption: Used for SecuBSP's window attribute

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_DEVICE_NOT_OPENED</i>	The device in an input parameter is not yet opened
<i>SecuAPIERROR_USER_CANCEL</i>	Use of "Cancel" button closed function

SecuAPI_Process

```
SecuAPI_RETURN SecuAPI_Process(
    IN          SecuAPI_HANDLE          Handle,
    IN          const SecuAPI_INPUT_FIR_PTR  CapturedFIR,
    OUT         SecuAPI_FIR_HANDLE_PTR    ProcessedFIR);
```

- **Description**

Processes the intermediate data captured via a call to *SecuAPI_Capture* for the purpose of either verification or identification. If the processing capability is in the current module, the module builds a "processed" FIR; otherwise, *ProcessedFIR* is set to NULL.

- **Parameters**

Handle: The handle of the SecuBSP module

CapturedFIR: A handle to a captured FIR

ProcessedFIR: A pointer to the newly constructed "processed" FIR

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
<i>SecuAPIERROR_ALREADY_PROCESSED</i>	The template was already processed
<i>SecuAPIERROR_EXTRACTION_OPEN_FAIL</i>	The extraction module cannot be opened
<i>SecuAPIERROR_DATA_PROCESS_FAIL</i>	Internal error occurred during data processing
<i>SecuAPIERROR_ENCRYPTED_DATA_ERROR</i>	The encrypted FIR data in an input parameter is broken
<i>SecuAPIERROR_INTERNAL_CHECKSUM_FAIL</i>	The checksum of FIR data in an input parameter is invalid

SecuAPI_CreateTemplate

```
SecuAPI_RETURN SecuAPI_CreateTemplate(
    IN          SecuAPI_HANDLE          Handle,
    IN          const SecuAPI_INPUT_FIR_PTR  CapturedFIR,
    IN          const SecuAPI_INPUT_FIR_PTR  StoredTemplate,
```

```

        OUT      SecuAPI_FIR_HANDLE_PTR      NewTemplate,
        IN        const SecuAPI_FIR_PAYLOAD_PTR Payload
    );

```

- **Description**

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

- **Parameters**

Handle: The handle of the SecuBSP module

CapturedFIR: The pointer to captured FIR. Only input CapturedFIR with the Purpose set to 'Enroll for verification' may be accepted.

StoredTemplate: The template to be adapted (optional)

NewTemplate: A handle to a newly created template that is derived from the CapturedFIR and (optionally) the StoredTemplate.

Payload: A pointer to data that will be wrapped inside the newly created template. This parameter is ignored if NULL.

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
<i>SecuAPIERROR_ENCRYPTED_DATA_ERROR</i>	The encrypted FIR data in an input parameter is broken
<i>SecuAPIERROR_INTERNAL_CHECKSUM_FAIL</i>	The checksum of FIR data in an input parameter is invalid
<i>SecuAPIERROR_MUST_BE_PROCESSED_DATA</i>	The function requires a fully processed FIR

SecuAPI_VerifyMatch

```

SecuAPI_RETURN SecuAPI_VerifyMatch(
    IN      SecuAPI_HANDLE      Handle,
    IN      const SecuAPI_INPUT_FIR_PTR      ProcessedFIR,
    IN      const SecuAPI_INPUT_FIR_PTR      StoredTemplate,
    OUT     SecuAPI_BOOL*       Result,
    OUT     SecuAPI_FIR_PAYLOAD_PTR      PayLoad
);

```

- **Description**

Performs a 1:1 (one-to-one) verification between the ProcessedFIR and the StoredTemplate. The ProcessedFIR is the "processed" FIR constructed specifically for this verification. The StoredTemplate was created at enrollment. If the StoredTemplate contains a Payload, the Payload may be returned upon successful verification.

- **Parameters**

Handle: The handle of the SecuBSP module

ProcessedFIR: The FIR to be verified

StoredTemplate: The stored FIR with which input FIR will be compared

Result: A pointer to a Boolean value (SecuAPI_TRUE/ SecuAPI_FALSE) indicating whether the FIRs match

or not, according to the specified criteria

Payload: If the StoredTemplate contains a payload, it is returned in an allocated SecuAPI_FIR_PAYLOAD structure if Result value is SecuAPI_TRUE.

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
<i>SecuAPIERROR_VERIFICATION_OPEN_FAIL</i>	The verification module cannot be opened
<i>SecuAPIERROR_ENCRYPTED_DATA_ERROR</i>	The encrypted FIR data in an input parameter is broken
<i>SecuAPIERROR_INTERNAL_CHECKSUM_FAIL</i>	The checksum of FIR data in an input parameter is invalid
<i>SecuAPIERROR_MUST_BE_PROCESSED_DATA</i>	The function requires a fully processed FIR

SecuAPI_Enroll

```

SecuAPI_RETURN SecuAPI_Enroll(
    IN          SecuAPI_HANDLE          Handle,
    IN          const SecuAPI_INPUT_FIR_PTR  StoredTemplate,
    OUT         SecuAPI_FIR_HANDLE_PTR    NewTemplate,
    IN          const SecuAPI_FIR_PAYLOAD_PTR Payload,
    IN          SecuAPI_SINT32           Timeout,
    OUT         SecuAPI_FIR_HANDLE_PTR    AuditData,
    IN          const SecuAPI_WINDOW_OPTION_PTR WindowOption
);

```

- **Description**

Captures fingerprint data from the attached reader to create a ProcessedFIR for the purpose of enrollment

- **Parameters**

Handle: The handle of the SecuBSP module

StoredTemplate: The template to be adapted (optional)

NewTemplate: A handle to a newly created template that is derived from the CapturedFIR and (optionally) the StoredTemplate

Payload: A pointer to data that will be wrapped inside the newly created template

Timeout: 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. If value is -1, then default value will be used.

AuditData: A handle to an FIR containing raw fingerprint image data. This data may be used to provide human-identifiable data specific to the person at the reader. If the pointer is NULL on input, no audit data is collected.

WindowOption: Used for SecuBSP's window attribute

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
<i>SecuAPIERROR_DEVICE_NOT_OPENED</i>	The device in an input parameter is not yet opened

<i>SecuAPIERROR_EXTRACTION_OPEN_FAIL</i>	The extraction module cannot be opened
<i>SecuAPIERROR_VERIFICATION_OPEN_FAIL</i>	The verification module cannot be opened
<i>SecuAPIERROR_ENCRYPTED_DATA_ERROR</i>	The encrypted FIR data in an input parameter is broken
<i>SecuAPIERROR_INTERNAL_CHECKSUM_FAIL</i>	The checksum of FIR data in an input parameter is invalid
<i>SecuAPIERROR_FUNCTION_FAIL</i>	Function failed for unknown reason (internal)
<i>SecuAPIERROR_USER_CANCEL</i>	Use of “Cancel” button closed function

SecuAPI_Verify

```

SecuAPI_RETURN SecuAPI_Verify (
    IN          SecuAPI_HANDLE          Handle,
    IN          const SecuAPI_INPUT_FIR_PTR  StoredTemplate,
    OUT         SecuAPI_BOOL*           Result,
    OUT         SecuAPI_FIR_PAYLOAD_PTR    Payload,
    IN          SecuAPI_SINT32          Timeout,
    OUT         SecuAPI_FIR_HANDLE_PTR    AuditData,
    IN          const SecuAPI_WINDOW_OPTION_PTR WindowOption
);

```

- **Description**

Captures fingerprint data from the attached reader, and compares it against the StoredTemplate

- **Parameters**

Handle: The handle of the SecuBSP module

StoredTemplate: The stored FIR with which input FIR will be compared

Result: Matching result

Payload: If the StoredTemplate contains a payload, it is returned in an allocated data structure.

Timeout: 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. If value is –1, then default value will be used.

AuditData: A handle to a FIR containing raw fingerprint image data. This data may be used to provide human-identifiable data of the person at the reader. If the pointer is NULL on input, no audit data is collected.

WindowOption: Used for SecuBSP's window attribute

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_HANDLE</i>	There is an invalid handle in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
<i>SecuAPIERROR_USER_CANCEL</i>	Use of “Cancel” button closed function
<i>SecuAPIERROR_ENCRYPTED_DATA_ERROR</i>	The encrypted FIR data in an input parameter is broken
<i>SecuAPIERROR_INTERNAL_CHECKSUM_FAIL</i>	The checksum of FIR data in an input parameter is invalid
<i>SecuAPIERROR_EXTRACTION_OPEN_FAIL</i>	The extraction module cannot be opened
<i>SecuAPIERROR_VERIFICATION_OPEN_FAIL</i>	The verification module cannot be opened

3.4. User Interface Functions

SecuAPI_SetSkinResource

```
SecuAPI_BOOL SecuAPI_SetSkinResource(
    IN          LPCTSTR          szResPath
);
```

- **Description**
Sets a new skin resource to BSP module (skin resources can be made for OEM users)
- **Parameters**
szResPath: Path of resource DLL. If this value is set to NULL, BSP uses default resource.
- **Returned Values**
SecuAPI_TRUE: The function was successful
SecuAPI_FALSE: Cannot find resource DLL. In this case, BSP uses default resource.

3.5. Utility Functions

SecuAPI_FDxToSecuBSP

```
SecuAPI_RETURN SecuAPI SecuAPI_FDxToSecuBSP(
    IN          SecuAPI_HANDLE          Handle,
    IN          SecuAPI_UINT8*          FDxData,
    IN          SecuAPI_UINT32          FDxDataSize,
    IN          SecuAPI_UINT32          FDxDataType,
    IN          SecuAPI_FIR_PURPOSE     Purpose,
    OUT         SecuAPI_FIR_HANDLE_PTR  ProcessedFIR
);
```

- **Description**
Converts FDx template format (400 byte minutiae array) to FIR format. Converted data will be FIR handle. This function does not take raw data format.
- **Parameters**
Handle: The handle of the SecuBSP module
FDxData: The array of FDx data
FDxDataSize: The size of FDx data in bytes (must be a multiple of 400. e.g., 400, 800, 1200, etc.)
FDxDataType: Specifies FDx data type
In SecuBSP SDK, the following types are supported:
MINCONV_TYPE_FDP = 0: Data captured from FDP01/FDP02-based reader
MINCONV_TYPE_FDU = 1: Data captured from FDU01/FDU02/FDU03/SDU03-based reader
MINCONV_TYPE_FDA = 2: Data captured from FDA02/SDA0x device
MINCONV_TYPE_OLD_FDA = 3: Data captured from FDA01 device

In SecuBSP SDK *Pro*, the following types are supported:
MINCONV_TYPE_SG400 = 11: Data captured from FDx SDK *Pro* or Stand-alone type device
MINCONV_TYPE_OLD_FDX = 12: Data captured from FDx SDK

Purpose: The Purpose of the processedFIR
- *SecuAPI_FIR_PURPOSE_ENROLL* – Data captured for enrollment
- *SecuAPI_FIR_PURPOSE_IDENTIFY* – Data captured for identification

- *SecuAPI_FIR_PURPOSE_VERIFY* – Data captured for verification
ProcessedFIR: A handle to an FIR containing processed data

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
<i>SecuAPIERROR_INVALID_MINSIZE</i>	The minutiae size is not valid
<i>SecuAPIERROR_FUNCTION_FAIL</i>	Conversion failed

SecuAPI_SecuBSPToFDx

```
SecuAPI_RETURN SecuAPI_SecuAPI_FDxFtoSecuBSP(
    IN          SecuAPI_HANDLE          Handle,
    IN          const SecuAPI_INPUT_FIR_PTR piFIR,
    OUT         SecuAPI_EXPORT_DATA_PTR  pExportData,
    IN          MINCONV_DATA_TYPE        nExportType
);
```

- **Description**

Exports FDx template format (400 byte minutiae array) from FIR format data. The converted data is contained in *SecuAPI_EXPORT_DATA* structure.

- **Parameters**

Handle: The handle of the SecuBSP module

piFIR: The FIR input to be converted

pExportData: The pointer of *SecuAPI_EXPORT_DATA* structure. This structure contains information about each fingerprint in FIR and its minutiae data.

FDxDataType: Specifies FDx data type

In SecuBSP SDK, the following types are supported:

MINCONV_TYPE_FDP = 0: Data captured from FDP01/FDP02-based reader

MINCONV_TYPE_FDU = 1: Data captured from FDU01/FDU02/FDU03/SDU03-based reader

MINCONV_TYPE_FDA = 2: Data captured from FDA02/SDA0x device

MINCONV_TYPE_OLD_FDA = 3: Data captured from FDA01 device

In SecuBSP SDK *Pro*, the following types are supported:

MINCONV_TYPE_SG400 = 11: Data captured from FDx SDK *Pro* or Stand-alone type device

MINCONV_TYPE_OLD_FDX = 12: Data captured from FDx SDK

- **Returned Values**

<i>SecuAPIERROR_NONE</i>	The function was successful
<i>SecuAPIERROR_INVALID_POINTER</i>	There is an invalid pointer in an input function parameter or input field of a data structure
<i>SecuAPIERROR_FUNCTION_FAIL</i>	Conversion failed

SecuAPI_ExportImage

```
SecuAPI_RETURN SecuAPI_SecuBSPToImage(
    IN          SecuAPI_HANDLE          Handle,
    IN          const SecuAPI_INPUT_FIR_PTR piFIR,
    OUT         SecuAPI_EXPORT_AUDIT_DATA_PTR pExportData
);
```

- **Description**

Exports image data from FIR audit data. The exported data information and image data are contained in the *SecuAPI_EXPORT_AUDIT_DATA* structure.

- **Parameters**

Handle: The handle of the SecuBSP module

piFIR: The Audit FIR to be converted

pExportData: The pointer of SecuAPI_EXPORT_AUDIT_DATA structure. This structure contains information about each fingerprint in FIR and its image data.

- **Returned Values**

SecuAPIERROR_NONE

The function was successful

SecuAPIERROR_INVALID_POINTER

There is an invalid pointer in an input function parameter or input field of a data structure

SecuAPIERROR_FUNCTION_FAIL

Conversion failed