SecuGen

SecuSearch® 3 Pro SDK Manual

Windows

For one-to-many fingerprint identification applications

Using SecuGen® fingerprint readers

SG1-0005D-005 (04/23)

About SecuGen

SecuGen (www.secugen.com) provides biometric products and development tools for development organizations that are creating physical and network security systems 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 developing 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 are renowned for their accuracy, reliability, ruggedness, and affordability. Based in Silicon Valley, SecuGen has been serving the global biometric community since 1998.

About SecuSearch® Pro

SecuSearch® Pro is a 1:N (also called one-to-many or 1:many) matching engine for the rapid identification of fingerprint templates from among a large database of templates. SecuSearch Pro can handle rapid searches of up to 2 million templates.

Fingerprint matching algorithms fall into two main categories, 1:1 Matching and 1:N Matching.

1:1 (One-to-One) Matching [available with FDx® Pro]

> In instances where security is a high priority, 1:1 matching is typically used. This method is more secure than 1:N because 1:1 matching requires the person to present information that identifies themselves, as described below.
>
> Using a 1:1 template matching algorithm is called "authentication." This means that something is already known about a person. This information can be a customer ID, name, driver's license number, etc., and it is stored with that person's fingerprint template in a database. Using that information, the programmer obtains that person's fingerprint template from the database and then attempts to match that template with a template created from a newly captured fingerprint. If they match, then the person's identity is said to be verified or authenticated.

1:N (One-to-Many) Matching [available with SecuSearch® Pro]

> In instances where convenience is a high priority, 1:N matching is typically used. This method is fast and requires no previous knowledge of the person.
>
> Using a 1:N template matching algorithm, such as **SecuSearch Pro**, is called "identification." In this situation, no information about the person is required for the matching process except that their fingerprint must have been previously enrolled in the SecuSearch Pro database. To identify a person, a new fingerprint is captured, and the template extracted from this fingerprint is compared to the fingerprint templates stored in the SecuSearch Pro database. If a match is found, then the person is said to be identified.

Contents

Chapter 1. Overview

SecuSearch® Pro is a 1:N matching engine for the rapid identification of fingerprint templates from among a large database of templates. SecuSearch Pro can handle rapid searches of up to 2 million templates. The SecuSearch Pro SDK is a software developer's kit that enables programmers to develop extremely fast, highly accurate fingerprint searching programs for use in large-scale fingerprint databases.

Fingerprint matching algorithms fall into two main categories, 1:1 Matching and 1:N Matching.

1:1 (One-to-One) Matching [available with FDx® Pro]

> In instances where security is a high priority, 1:1 matching is typically used. This method is more secure than 1:N because 1:1 matching requires the person to present information that identifies themselves, as described below.
>
> Using a 1:1 template matching algorithm is called "authentication." This means that something is already known about a person. This information can be a customer ID, name, driver's license number, etc., and it is stored with that person's fingerprint template in a database. Using that information, the programmer obtains that person's fingerprint template from the database and then attempts to match that template with a template created from a newly captured fingerprint. If they match, then the person's identity is said to be verified or authenticated.

1:N (One-to-Many) Matching [available with SecuSearch® Pro]

> In instances where convenience is a high priority, 1:N matching is typically used. This method is fast and requires no previous knowledge of the person.
>
> Using a 1:N template matching algorithm, such as **SecuSearch Pro**, is called "identification." In this situation, no information about the person is required for the matching process except that their fingerprint must have been previously enrolled in the SecuSearch Pro database. To identify a person, a new fingerprint is captured, and the template extracted from this fingerprint is compared to the fingerprint templates stored in the SecuSearch Pro database. If a match is found, then the person is said to be identified.

SecuSearch Pro SDK can be used for two typical classes of applications:

- To identify unknown individuals by matching fingerprints against a fingerprint database (e.g., searching for missing children, criminal investigations, etc.)
- To identify registered users without requiring identifying information such as ID codes (e.g., time and attendance systems, member management systems, system login without ID, etc.)

SecuSearch Pro SDK supports quick and easy 1:N matching system integration in any fingerprint database application where accuracy and search speed are paramount.

1.1. Features of SecuSearch SDK

- **Optimized API**

  Offers an optimized API for fingerprint registration and searching so that programmers can quickly and easily build fingerprint search systems

- **High-speed fingerprint searching**

  Utilizes an innovative indexing-based algorithm that is different from sequential comparison and that increases the search speed to over 500,000 fingerprints per second in a PC environment

- **High accuracy in fingerprint matching**

  Provides accurate matching candidate lists with corresponding matching confidence levels

- **Wide compatibility**

  Can be easily integrated into any type of platform since it has been built in the ANSI C/C++ environment and currently supports 64 bit versions of Windows 10/8/7/2012R2.

1.2. SecuSearch API Functions

SecuSearch Engine uses the SG400 fingerprint template data extracted by SecuGen FDx SDK Pro APIs. Registration and search functions use the fingerprint template obtained from the image captured by SecuGen fingerprint readers. All functionalities of SecuSearch are encapsulated in the SecuSearchAPI C++ class.

SG400 is SecuGen’s proprietary fingerprint data format for storing features extracted from a fingerprint.

Main functions of SecuSearchAPI Class

Initialization, Termination and Configuration

- **InitializeEngine**

  Initializes the Fingerprint Database, allocates database memory, and initializes global variables

- **TerminateEngine**

  Clears the Fingerprint Database, deletes all fingerprint information located in the database, and frees allocated memory

- **GetEngineParam**

Retrieves values of SecuSearch Engine parameters

Fingerprint Registration, Removal, Searching and Identification

- **RegisterFP**

  Registers a new fingerprint template to the database

- **RegisterFPBatch**

  Registers new fingerprint templates to the database in a batch to improve the registration speed

- **RemoveFP**

  Deletes a fingerprint template from the database

- **RemoveFPBatch**

  Deletes a batch of fingerprint templates from the database

- **SearchFP**

  Searches the fingerprints in the database and outputs a list of matching candidates. Matching candidates are the fingerprints in the database that show similarity to the query fingerprint.

- **IdentifyFP**

  Identifies the fingerprints in the database and outputs the fingerprint that is most similar to the input (query) fingerprint, that is, a fingerprint in the database having a matching confidence level that's higher than a specified security level or threshold

- **ClearFPDB**

  Deletes all fingerprint templates from the memory resident fingerprint database

- **GetFPCount**

  Retrieves number of fingerprint templates registered in the memory resident fingerprint database

- **GetIDList**

  Retrieves a list of fingerprint templates registered in the memory resident fingerprint database

- **GetTemplate**

  Retrieves fingerprint template data registered in the memory resident fingerprint database

Database Management

- **SaveFPDB**

  Saves a memory resident fingerprint database to the disk

- **LoadFPDB**

  Loads a fingerprint database file from the disk to the memory resident DB

Standard Template Conversion

- **ExtractTemplate**

  Extracts a finger view from a standard template data of ANSI 378 or ISO 19794, and then converts it into SecuGen proprietary template data, which always includes a single view

- **GetNumberOfView**

  Gets the total number of views in an ANSI 378 or ISO 19794 template

Chapter 2. Installation

2.1. System Requirements

OS	Windows 10 / 8.1 / 8 / 7
	Windows Server 2016 / 2012 R2 / 2012 / 2008 R2 / 2008
CPU	Intel Core i7 3.4GHz or higher
System Memory	64-bit Engine: 16GB or larger (32GB Recommended)
	32-bit Engine: 4GB or larger (8GB Recommended)
Development Env.	Microsoft Visual Studio 2012
Language Interfaces	C/C++ and .Net

SecuSearch Engine is designed to operate with SecuGen fingerprint readers and SecuGen fingerprint feature-extraction algorithms provided in SecuGen FDx SDK Pro software developer kits.

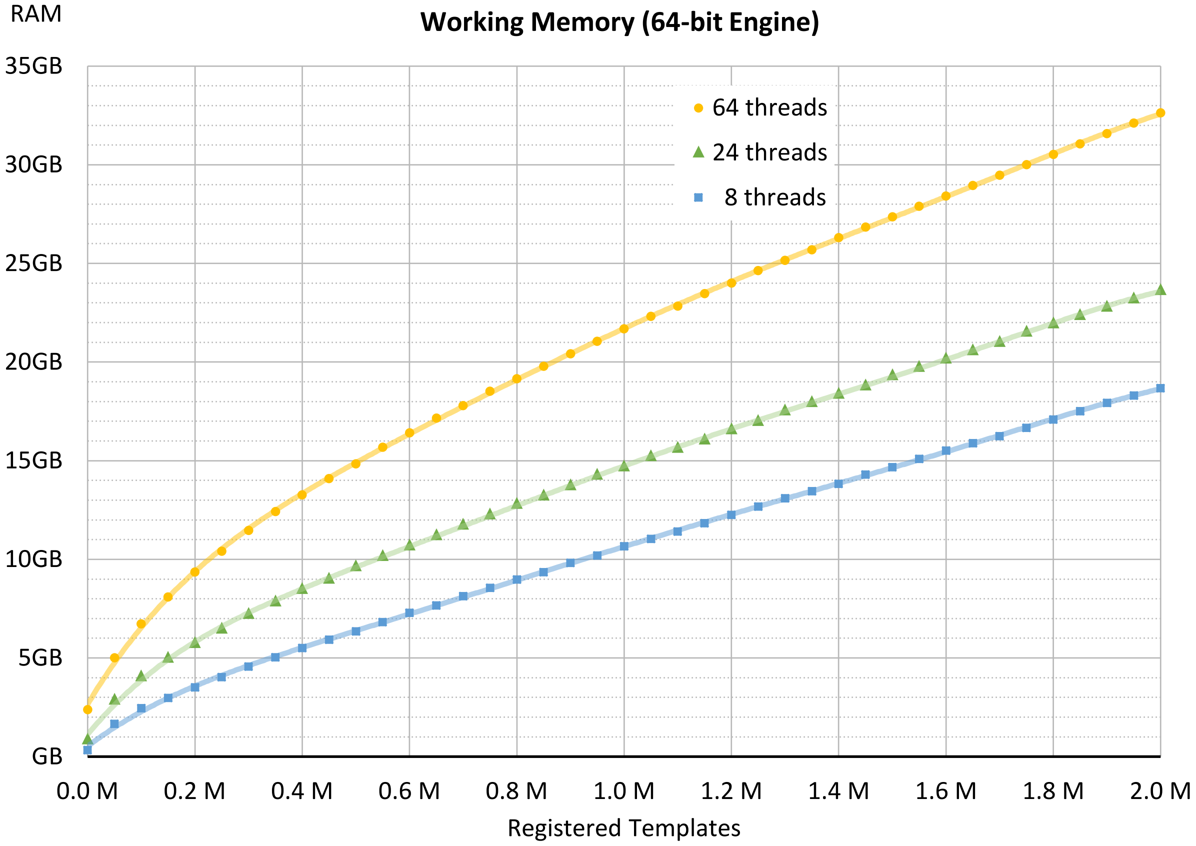
Memory requirements of SecuSearch Engine vary according to the number of registered fingerprints because the fingerprint database resides on the system's main memory (RAM).

Please note that it is required for the system memory to be much larger than the working memory of the SecuSearch Engine because it requires a large amount of contiguous memory block, and running other applications may reduce the maximum size of contiguous memory available.

***Important**: SecuSearch 3 32-bit engine supports up to 100,000 templates due to the 2GB memory constraint of 32-bit applications.*

2.1.1. Working Memory of 64-bit Engine

The following graph and table show the minimum working memory of the SecuSearch Engine 64-bit according to various counts of registered templates and the concurrency values (thread count) as its engine parameter.



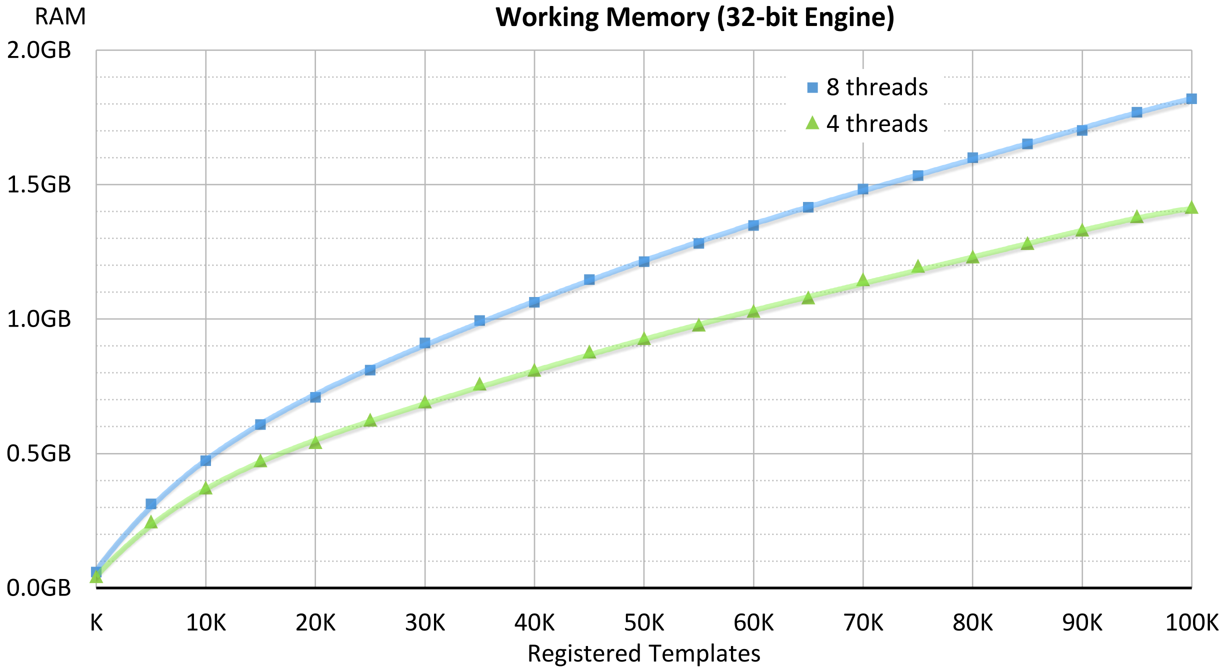
Working Memory (64-bit Engine)

Templates	8 threads	24 threads	64 threads
0	0.3 GB	0.9 GB	2.4 GB
50K	1.7 GB	2.9 GB	5.0 GB
100K	2.5 GB	4.1 GB	6.7 GB
200K	3.5 GB	5.7 GB	9.4 GB
300K	4.6 GB	7.2 GB	11.5 GB
400K	5.5 GB	8.5 GB	13.3 GB
500K	6.3 GB	9.6 GB	14.8 GB
600K	7.3 GB	10.7 GB	16.4 GB
700K	8.1 GB	11.7 GB	17.8 GB
800K	9.0 GB	12.8 GB	19.2 GB

900K	9.8 GB	13.7 GB	20.4 GB
1,000K	10.7 GB	14.7 GB	21.7 GB
1,100K	11.4 GB	15.6 GB	22.8 GB
1,200K	12.2 GB	16.6 GB	24.0 GB
1,300K	13.1 GB	17.5 GB	25.2 GB
1,400K	13.8 GB	18.4 GB	26.3 GB
1,500K	14.7 GB	19.3 GB	27.4 GB
1,600K	15.5 GB	20.2 GB	28.4 GB
1,700K	16.2 GB	21.0 GB	29.5 GB
1,800K	17.1 GB	22.0 GB	30.5 GB
1,900K	17.9 GB	22.8 GB	31.6 GB
2,000K	18.7 GB	23.6 GB	32.6 GB

2.1.2. Working Memory of 32-bit Engine

The following graph and table show the minimum working memory of the SecuSearch Engine 32-bit according to various counts of registered templates and the concurrency values (thread count) as its engine parameter.



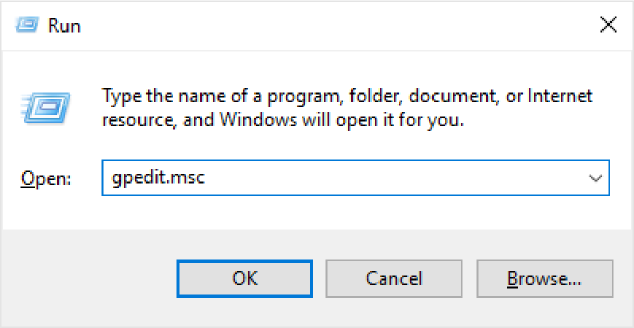
Working Memory (32-bit Engine)

Templates	4 threads	8 threads
0	39 MB	59 MB
5K	241 MB	313 MB
10K	368 MB	474 MB
15K	469 MB	609 MB
20K	536 MB	709 MB
25K	620 MB	811 MB
30K	688 MB	911 MB
35K	755 MB	996 MB
40K	806 MB	1.06 GB
45K	873 MB	1.15 GB
50K	924 MB	1.22 GB
55K	974 MB	1.28 GB
60K	1.03 GB	1.35 GB
65K	1.08 GB	1.42 GB
70K	1.14 GB	1.49 GB
75K	1.19 GB	1.53 GB
80K	1.23 GB	1.60 GB
85K	1.28 GB	1.65 GB
90K	1.33 GB	1.70 GB
95K	1.38 GB	1.78 GB
100K	1.41 GB	1.82 GB

2.2. Windows Memory Policy Configuration

SecuSearch Engine requires contiguous memory to be locked to function correctly. The Windows system policy must be configured as shown.

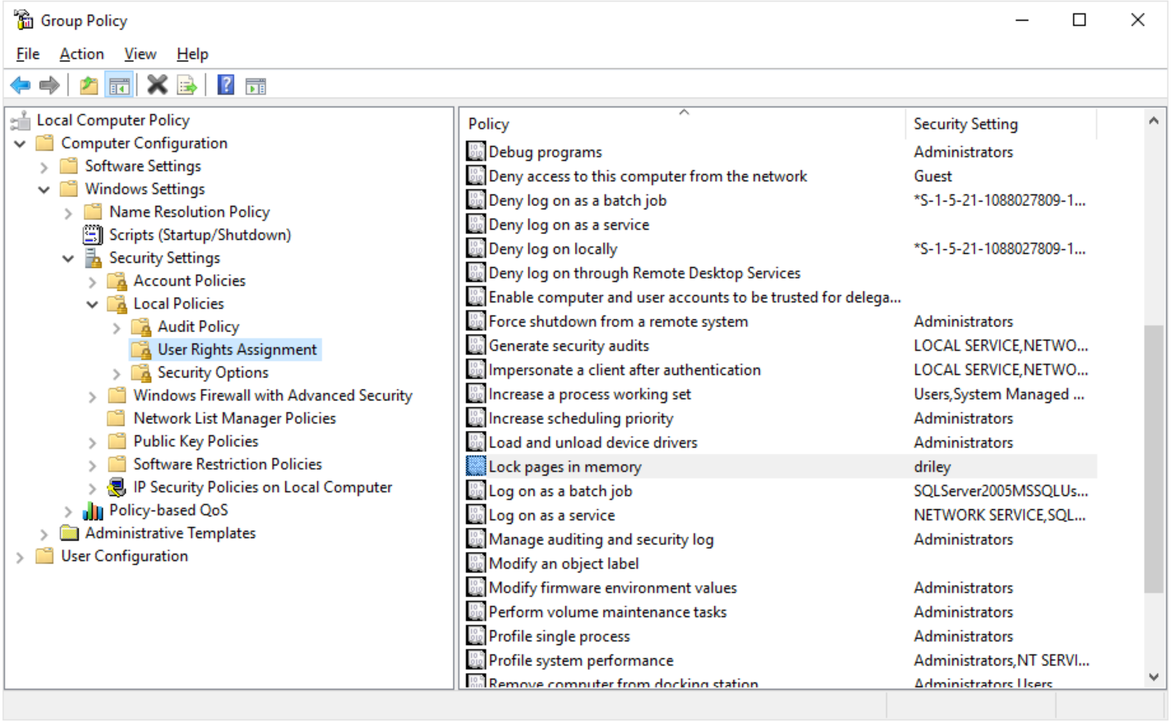
1. Launch ***gpedit.msc***.



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: gpedit.msc

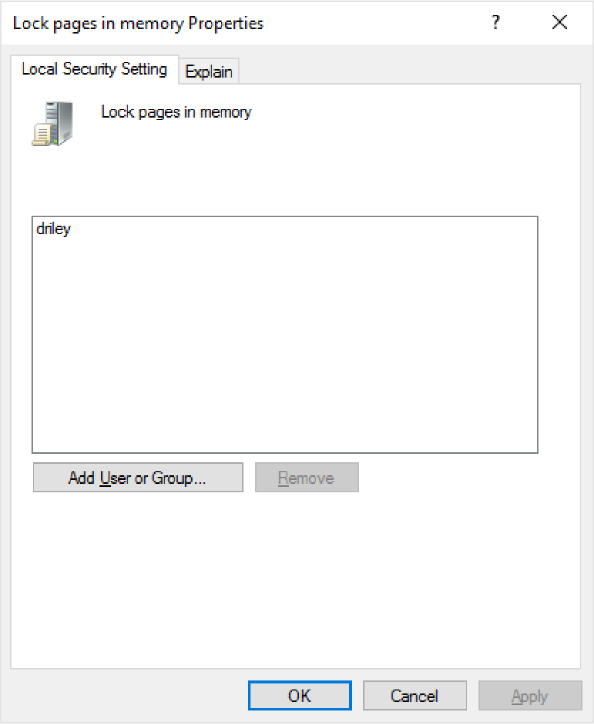
2. Navigate to ***Computer Configuration-> Windows Settings -> Security Settings -> User Rights Assignment***.
3. Double-click "***Lock pages in memory***" and add the user or group that will be running the SecuSearch Engine.



- Local Computer Policy
  - Computer Configuration
    - Software Settings
    - Windows Settings
      - Name Resolution Policy
      - Scripts (Startup/Shutdown)
      - Security Settings
        - Account Policies
        - Local Policies
          - Audit Policy
          - User Rights Assignment
          - Security Options
        - Windows Firewall with Advanced Security
        - Network List Manager Policies
        - Public Key Policies
        - Software Restriction Policies
        - IP Security Policies on Local Computer
      - Policy-based QoS
    - Administrative Templates
  - User Configuration

Policy	Security Setting
Debug programs	Administrators
Deny access to this computer from the network	Guest
Deny log on as a batch job	*S-1-5-21-1088027809-1...
Deny log on as a service	
Deny log on locally	*S-1-5-21-1088027809-1...
Deny log on through Remote Desktop Services	
Enable computer and user accounts to be trusted for delega...	
Force shutdown from a remote system	Administrators
Generate security audits	LOCAL SERVICE,NETWO...
Impersonate a client after authentication	LOCAL SERVICE,NETWO...
Increase a process working set	Users,System Managed ...
Increase scheduling priority	Administrators
Load and unload device drivers	Administrators
Lock pages in memory	driley
Log on as a batch job	SQLServer2005MSSQLUs...
Log on as a service	NETWORK SERVICE,SQL...
Manage auditing and security log	Administrators
Modify an object label	
Modify firmware environment values	Administrators
Perform volume maintenance tasks	Administrators
Profile single process	Administrators
Profile system performance	Administrators,NT SERVI...
Remove computer from docking station	Administrators,Users

4. Add the user or group that will be running the SecuSearch application by selecting "***Add User or Group…***". Only users with this privilege can start the SecuSearch engine.

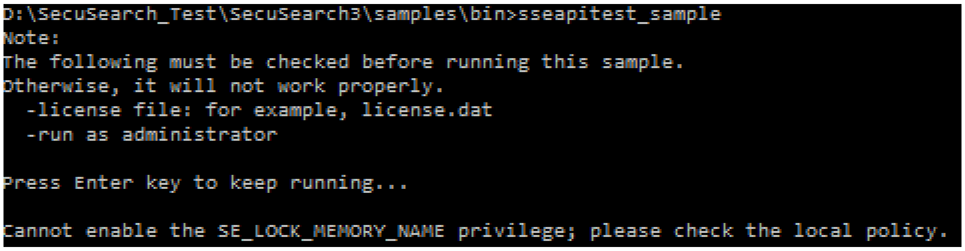


Lock pages in memory Properties

Lock pages in memory

drley

5. Reboot the system
6. If this is not done, you will receive an SE_LOCK_MEMORY_NAME error when running the SecuSearch sample application as show below:



```
D:\SecuSearch_Test\SecuSearch3\samples\bin>sseapitest_sample
Note:
The following must be checked before running this sample.
Otherwise, it will not work properly.
  -license file: for example, license.dat
  -run as administrator

Press Enter key to keep running...

Cannot enable the SE_LOCK_MEMORY_NAME privilege; please check the local policy.
```

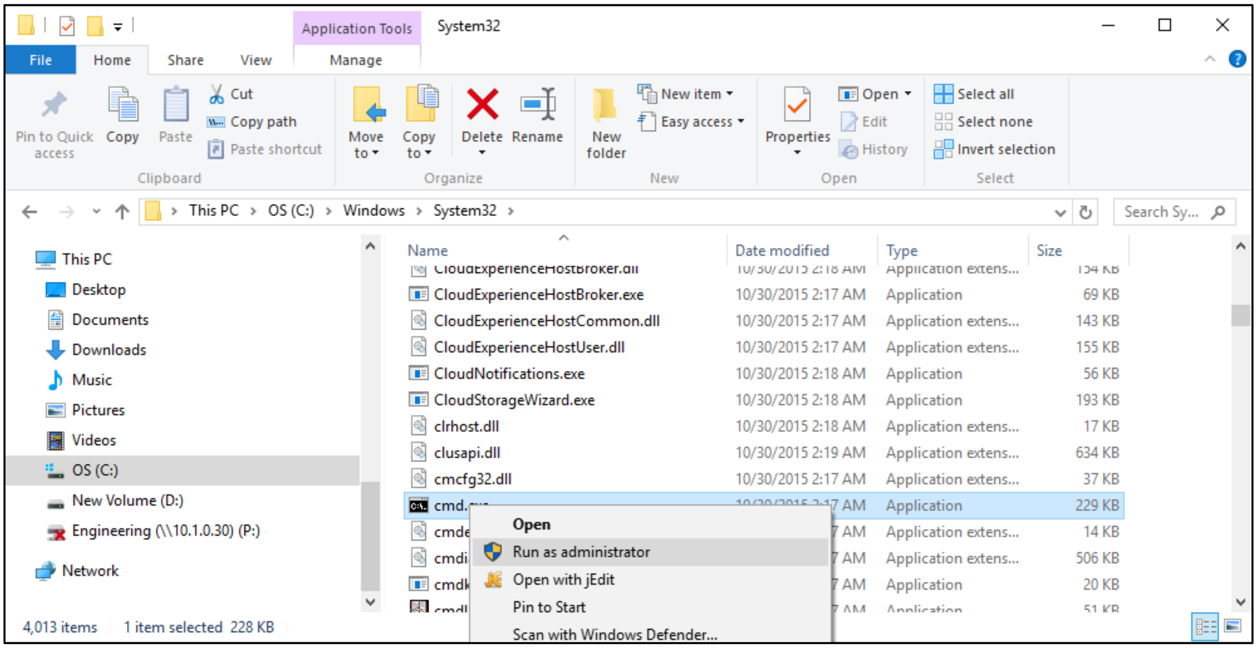
✓ For Web Environments

For web environments, running the SecuSearch engine in IIS, you should specify additional user or group settings for "Lock pages in memory" as shown above. In addition, you should add the IIS_IUSER group.

2.3. Administrative Privileges Required

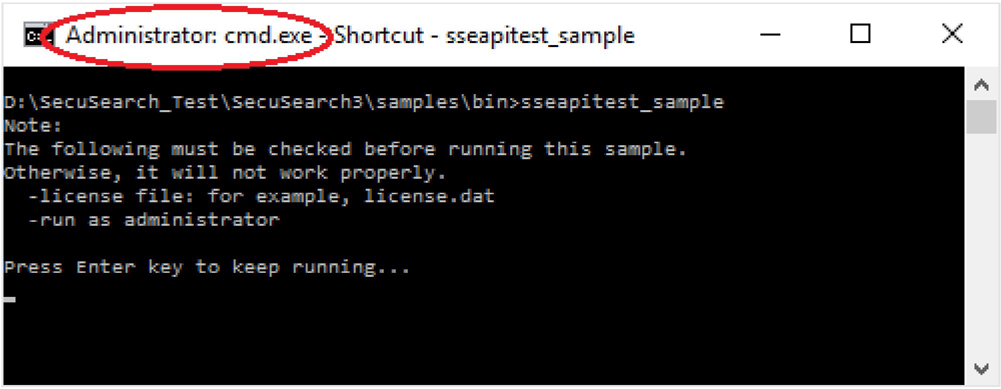
All applications built using SecuSearch Pro SDK must be launched with elevated privileges. When launching a command prompt to run the included sample applications, take the following steps:

1. Right click c:\windows\system32\cmd.exe and select "***Run as administrator***"



Name	Date modified	Type	Size
CloudExperienceHostBroker.dll	10/30/2015 2:18 AM	Application extens...	134 KB
CloudExperienceHostBroker.exe	10/30/2015 2:17 AM	Application	69 KB
CloudExperienceHostCommon.dll	10/30/2015 2:17 AM	Application extens...	143 KB
CloudExperienceHostUser.dll	10/30/2015 2:17 AM	Application extens...	155 KB
CloudNotifications.exe	10/30/2015 2:18 AM	Application	56 KB
CloudStorageWizard.exe	10/30/2015 2:18 AM	Application	193 KB
clrhost.dll	10/30/2015 2:18 AM	Application extens...	17 KB
clusapi.dll	10/30/2015 2:19 AM	Application extens...	634 KB
cmcfg32.dll	10/30/2015 2:17 AM	Application extens...	37 KB
cmd.exe	10/30/2015 2:17 AM	Application	229 KB
cmde	7 AM	Application extens...	14 KB
cmdi	7 AM	Application extens...	506 KB
cmdk	7 AM	Application	20 KB
cmdl	7 AM	Application	51 KB

2. The command prompt will be launched with administrative privileges as shown in the screen shot below:


Administrator: cmd.exe - Shortcut - sseapitest_sample

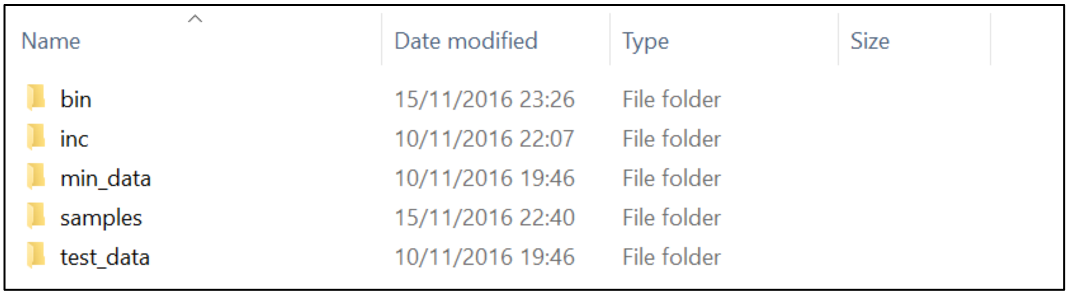
```
D:\SecuSearch_Test\SecuSearch3\samples\bin>sseapitest_sample
Note:
The following must be checked before running this sample.
Otherwise, it will not work properly.
  -license file: for example, license.dat
  -run as administrator

Press Enter key to keep running...
_
```

3. The included sample applications will now be run with elevated privileges when executed from the command line in this session.

✓ If you would like to debug your application together with SecuSearch engine in the Visual Studio IDE, you should execute the Visual Studio with administrative privileges.

2.4. SecuSearch SDK Pro Directory Structure

The SecuSearch SDK directory structure contains the following folders.



Name	Date modified	Type	Size
bin	15/11/2016 23:26	File folder	
inc	10/11/2016 22:07	File folder	
min_data	10/11/2016 19:46	File folder	
samples	15/11/2016 22:40	File folder	
test_data	10/11/2016 19:46	File folder	

1. **\bin: SecuSearch libraries and executable files**

   license.dat – License file that must be replaced by a SecuGen-issued file for your system

   SecuSearchAPI.dll – SecuSearch C/C++ API library

   SecuSearchAPI.Net.dll – SecuSearch .NET API library

   sseapitest_cpp.exe – SecuSearch C++ sample application

   sseapitest_csharp.exe – SecuSearch C# .NET sample application

   VolNoReader.exe – SecuSearch volume ID tool

2. **\inc: Header file**

   SecuSearchAPI.hpp – Defines SecuSearch C++ interface APIs in the SecuSearch SDK

   SecuSearchAPI.h – Defines SecuSearch C interface APIs in the SecuSearch SDK

   SecuSearchDef.h – Defines the basic data type used in the SecuSearch SDK and error values (included in SecuSearchAPI.h)

   SecuSearchErr.h – Defines error codes returned by SecuSearch interface APIs

3. **\min_data: Fingerprint minutiae files**

   These files are used with the sseapitest_sample.exe demo application included with the SDK.

4. **\samples: Sample Source code**

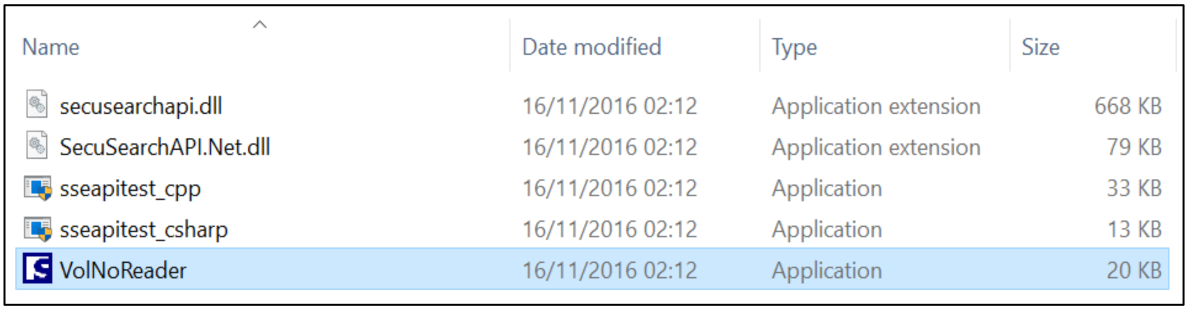
   This directory contains source codes for the sseapitest_cpp.exe and sseapitest_csharp.exe demo included with the SDK.

5. **\ test_data: ANSI/ISO Standard Template files**

2.5. Obtaining a SecuSearch License

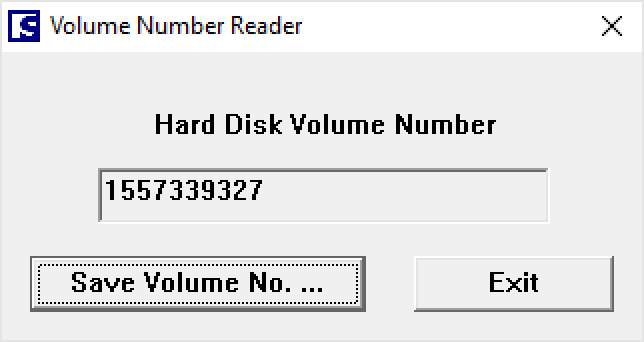
SecuSearch database is built to hold 1,000 templates without a license file. If the SecuSearch database needs to have more than 1,000 templates, then a license file generated by SecuGen Corporation will need to be obtained. Proceed as follows to obtain a valid license:

1. Launch the ***VolNoReader.exe*** application included with the SDK distribution.



Name	Date modified	Type	Size
secusearchapi.dll	16/11/2016 02:12	Application extension	668 KB
SecuSearchAPI.Net.dll	16/11/2016 02:12	Application extension	79 KB
sseapitest_cpp	16/11/2016 02:12	Application	33 KB
sseapitest_csharp	16/11/2016 02:12	Application	13 KB
VolNoReader	16/11/2016 02:12	Application	20 KB

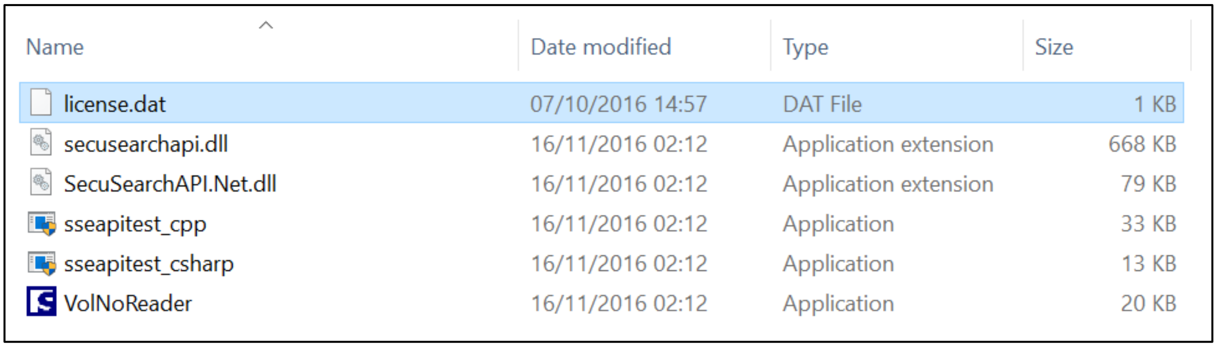
2. The volume number reader will display the volume ID of your system as shown below:



Hard Disk Volume Number

1557339327

3. Contact your SecuGen representative and provide the following information:
   a. System volume number
   b. Maximum number of fingerprints to be loaded into SecuSearch
4. You will receive a license file from SecuGen as shown below. The path to this file should be passed to the SecuSearch ***SSE_PARAM*** structure that is used by the **InitializeEngine()** function.



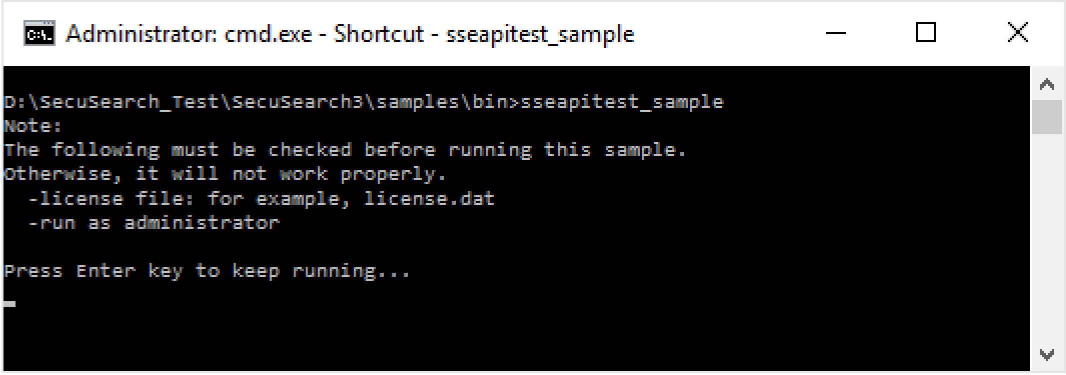
Name	Date modified	Type	Size
license.dat	07/10/2016 14:57	DAT File	1 KB
secusearchapi.dll	16/11/2016 02:12	Application extension	668 KB
SecuSearchAPI.Net.dll	16/11/2016 02:12	Application extension	79 KB
sseapitest_cpp	16/11/2016 02:12	Application	33 KB
sseapitest_csharp	16/11/2016 02:12	Application	13 KB
VolNoReader	16/11/2016 02:12	Application	20 KB

Chapter 3. Using Sample Programs

3.1. SecuSearch API Test Sample

This sample application is located in the bin directory of the SDK distribution. It references fingerprint data stored in the *min_data* directory of the SDK distribution.

1. Configure your Windows memory policy as detailed in Section 2.2.
2. Launch a command prompt with administrative privileges as described in Section 2.3.
3. Navigate to the bin directory of the SecuSearch SDK distribution and execute ***sseapitest_sample.exe*** as shown below.



```
D:\SecuSearch_Test\SecuSearch3\samples\bin>sseapitest_sample
Note:
The following must be checked before running this sample.
Otherwise, it will not work properly.
  -license file: for example, license.dat
  -run as administrator

Press Enter key to keep running...
_
```

4. Press <Enter> when prompted
5. Upon successful execution, the output will be as follows:

Note:

The following must be checked before running this sample. Otherwise, it will not work properly.

- ✓ license file: for example, license.dat
- ✓ run as administrator

```
Note:
The following must be checked before running this sample.
Otherwise, it will not work properly.
  -license file: for example, license.dat
  -run as administrator

Press Enter key to keep running...

API version: 3.2.2
```

```
loading template files in ../min_data
total db size = 51
1 adding 200000_0_0.min
2 adding 200001_0_0.min
3 adding 200002_0_0.min
4 adding 200003_0_0.min
5 adding 200004_0_0.min
6 adding 200005_0_0.min
7 adding 200006_0_0.min
8 adding 200007_0_0.min
9 adding 200008_0_0.min
10 adding 200009_0_0.min
11 adding 200010_0_0.min
12 adding 200011_0_0.min
13 adding 200012_0_0.min
14 adding 200013_0_0.min
15 adding 200014_0_0.min
16 adding 200015_0_0.min
17 adding 200016_0_0.min
18 adding 200017_0_0.min
19 adding 200018_0_0.min
20 adding 200019_0_0.min
21 adding 200020_0_0.min
22 adding 200021_0_0.min
23 adding 200022_0_0.min
24 adding 200023_0_0.min
25 adding 200024_0_0.min
26 adding 200025_0_0.min
27 adding 200026_0_0.min
28 adding 200027_0_0.min
29 adding 200028_0_0.min
30 adding 200029_0_0.min
31 adding 200030_0_0.min
32 adding 200031_0_0.min
33 adding 200032_0_0.min
34 adding 200033_0_0.min
35 adding 200034_0_0.min
36 adding 200035_0_0.min
37 adding 200036_0_0.min
38 adding 200037_0_0.min
39 adding 200038_0_0.min
40 adding 200039_0_0.min
41 adding 200040_0_0.min
42 adding 200041_0_0.min
43 adding 200042_0_0.min
44 adding 200043_0_0.min
45 adding 200044_0_0.min
```

```
46 adding 200045_0_0.min
47 adding 200046_0_0.min
48 adding 200047_0_0.min
49 adding 200048_0_0.min
50 adding 200049_0_0.min
51 adding 200050_0_0.min
mdb size = 51
loading templates done
Registering templates ...>>>
Registering templates done >>  fpcount = 51

write DB to sample.tdb
id = 0
id = 1
id = 2
id = 3
id = 4
id = 5
id = 6
id = 7
id = 8
id = 9
id = 10
id = 11
id = 12
id = 13
id = 14
id = 15
id = 16
id = 17
id = 18
id = 19
id = 20
id = 21
id = 22
id = 23
id = 24
id = 25
id = 26
id = 27
id = 28
id = 29
id = 30
id = 31
id = 32
id = 33
id = 34
id = 35
id = 36
id = 37
id = 38
id = 39
id = 40
id = 41
id = 42
id = 43
id = 44
id = 45
id = 46
id = 47
id = 48
id = 49
id = 50
Template DB is cleared >>  fpcount = 0
200000_0_0.min :    0 :  no candidate
200001_0_0.min :    1 :  no candidate
200002_0_0.min :    2 :  no candidate
200003_0_0.min :    3 :  no candidate
200004_0_0.min :    4 :  no candidate
200005_0_0.min :    5 :  no candidate
200006_0_0.min :    6 :  no candidate
200007_0_0.min :    7 :  no candidate
200008_0_0.min :    8 :  no candidate
200009_0_0.min :    9 :  no candidate
200010_0_0.min :   10 :  no candidate
200011_0_0.min :   11 :  no candidate
200012_0_0.min :   12 :  no candidate
200013_0_0.min :   13 :  no candidate
200014_0_0.min :   14 :  no candidate
200015_0_0.min :   15 :  no candidate
200016_0_0.min :   16 :  no candidate
200017_0_0.min :   17 :  no candidate
200018_0_0.min :   18 :  no candidate
200019_0_0.min :   19 :  no candidate
200020_0_0.min :   20 :  no candidate
200021_0_0.min :   21 :  no candidate
200022_0_0.min :   22 :  no candidate
200023_0_0.min :   23 :  no candidate
200024_0_0.min :   24 :  no candidate
200025_0_0.min :   25 :  no candidate
200026_0_0.min :   26 :  no candidate
200027_0_0.min :   27 :  no candidate
200028_0_0.min :   28 :  no candidate
200029_0_0.min :   29 :  no candidate
200030_0_0.min :   30 :  no candidate
200031_0_0.min :   31 :  no candidate
```

```
200032_0_0.min :   32 :  no candidate
200033_0_0.min :   33 :  no candidate
200034_0_0.min :   34 :  no candidate
200035_0_0.min :   35 :  no candidate
200036_0_0.min :   36 :  no candidate
200037_0_0.min :   37 :  no candidate
200038_0_0.min :   38 :  no candidate
200039_0_0.min :   39 :  no candidate
200040_0_0.min :   40 :  no candidate
200041_0_0.min :   41 :  no candidate
200042_0_0.min :   42 :  no candidate
200043_0_0.min :   43 :  no candidate
200044_0_0.min :   44 :  no candidate
200045_0_0.min :   45 :  no candidate
200046_0_0.min :   46 :  no candidate
200047_0_0.min :   47 :  no candidate
200048_0_0.min :   48 :  no candidate
200049_0_0.min :   49 :  no candidate
200050_0_0.min :   50 :  no candidate
Registering templates in batches ...>>>
Registering templates in batches done >>  fpcount = 51
200000_0_0.min :    0 :      0   matchScore=9999
200001_0_0.min :    1 :      1   matchScore=9999
200002_0_0.min :    2 :      2   matchScore=9999
200003_0_0.min :    3 :      3   matchScore=9999
200004_0_0.min :    4 :      4   matchScore=9999
200005_0_0.min :    5 :      5   matchScore=9999
200006_0_0.min :    6 :      6   matchScore=9999
200007_0_0.min :    7 :      7   matchScore=9999
200008_0_0.min :    8 :      8   matchScore=9999
200009_0_0.min :    9 :      9   matchScore=9999
200010_0_0.min :   10 :     10   matchScore=9999
200011_0_0.min :   11 :     11   matchScore=9999
200012_0_0.min :   12 :     12   matchScore=9999
200013_0_0.min :   13 :     13   matchScore=9999
200014_0_0.min :   14 :     14   matchScore=9999
200015_0_0.min :   15 :     15   matchScore=9999
200016_0_0.min :   16 :     16   matchScore=9999
200017_0_0.min :   17 :     17   matchScore=9999
200018_0_0.min :   18 :     18   matchScore=9999
200019_0_0.min :   19 :     19   matchScore=9999
200020_0_0.min :   20 :     20   matchScore=9999
200021_0_0.min :   21 :     21   matchScore=9999
200022_0_0.min :   22 :     22   matchScore=9999
200023_0_0.min :   23 :     23   matchScore=9999
200024_0_0.min :   24 :     24   matchScore=9999
200025_0_0.min :   25 :     25   matchScore=9999
200026_0_0.min :   26 :     26   matchScore=9999
```

```
200027_0_0.min :   27 :    27   matchScore=9999
200028_0_0.min :   28 :    28   matchScore=9999
200029_0_0.min :   29 :    29   matchScore=9999
200030_0_0.min :   30 :    30   matchScore=9999
200031_0_0.min :   31 :    31   matchScore=9999
200032_0_0.min :   32 :    32   matchScore=9999
200033_0_0.min :   33 :    33   matchScore=9999
200034_0_0.min :   34 :    34   matchScore=9999
200035_0_0.min :   35 :    35   matchScore=9999
200036_0_0.min :   36 :    36   matchScore=9999
200037_0_0.min :   37 :    37   matchScore=9999
200038_0_0.min :   38 :    38   matchScore=9999
200039_0_0.min :   39 :    39   matchScore=9999
200040_0_0.min :   40 :    40   matchScore=9999
200041_0_0.min :   41 :    41   matchScore=9999
200042_0_0.min :   42 :    42   matchScore=9999
200043_0_0.min :   43 :    43   matchScore=9999
200044_0_0.min :   44 :    44   matchScore=9999
200045_0_0.min :   45 :    45   matchScore=9999
200046_0_0.min :   46 :    46   matchScore=9999
200047_0_0.min :   47 :    47   matchScore=9999
200048_0_0.min :   48 :    48   matchScore=9999
200049_0_0.min :   49 :    49   matchScore=9999
200050_0_0.min :   50 :    50   matchScore=9999

Register and search an ANSI 378 template
registered the 0-th view
searched the 0-th view
expected candidate: 1234560
candidate: 1234560, matchScore=9999
registered the 1-th view
searched the 1-th view
expected candidate: 1234561
candidate: 1234561, matchScore=9999

Register and search an ISO 19794 template
registered the 0-th view
searched the 0-th view
expected candidate: 9876540
candidate: 9876540, matchScore=9999

Application is done. Press Enter key to close.
```

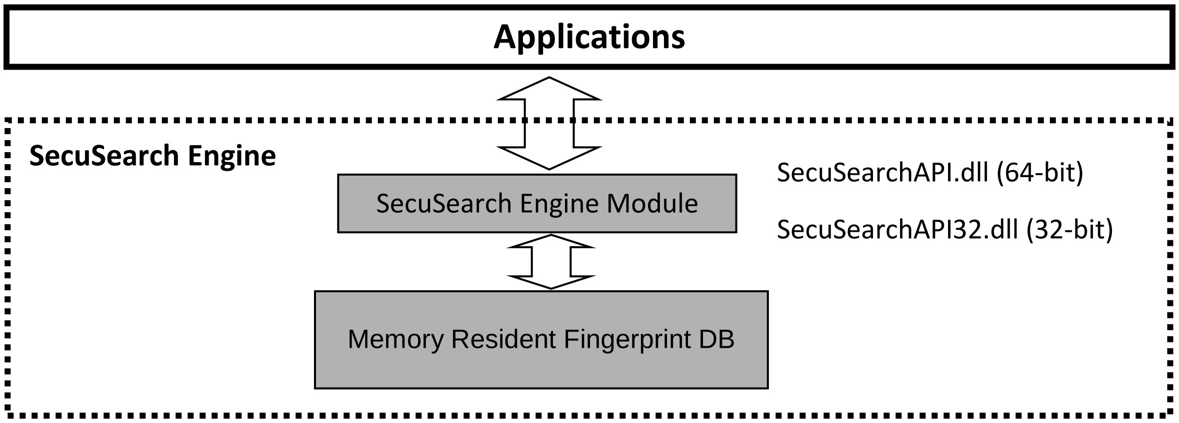
Chapter 4. SDK Programming

This chapter describes how to program with the one-to-many (1:N) identification system using the SecuSearch Pro SDK.

4.1. SecuSearch Engine Structure

The SecuSearch Engine uses as input the 400-byte fingerprint template format (SG400) that is generated by the SecuGen FDx SDK Pro.

SecuSearch Engine Structure (400-byte template format)


Applications
SecuSearch Engine
SecuSearch Engine Module
SecuSearchAPI.dll (64-bit)
SecuSearchAPI32.dll (32-bit)
Memory Resident Fingerprint DB

Module description

- **SecuSearch Engine Module**

  Main module that performs fingerprint registration, removal, and search operations as well as internal database management

- **Memory resident Fingerprint DB**

  Internal memory resident database that contains registered fingerprint information. The input templates are preprocessed with additional information to facilitate fast searching.

4.2. System Architecture

In this section, block diagrams are shown for each 1:N matching system used with SecuGen fingerprint recognition products.

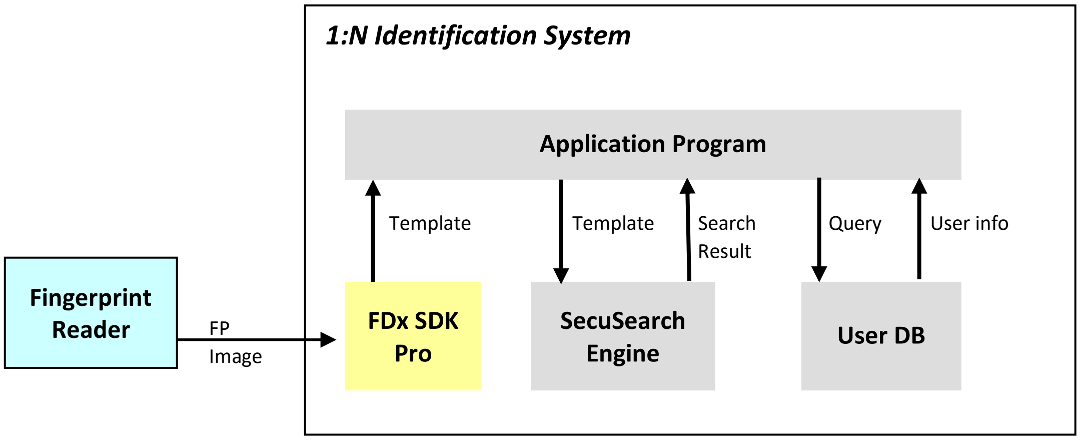
SecuGen fingerprint recognition products that work with the SecuSearch Engine:

SecuGen FDx SDK Pro: SecuGen low-level fingerprint recognition SDK that is used with SecuGen fingerprint peripherals. This SDK may be used to capture fingerprints and extract minutiae into templates (SG400 format) that can then be registered in the SecuSearch database.

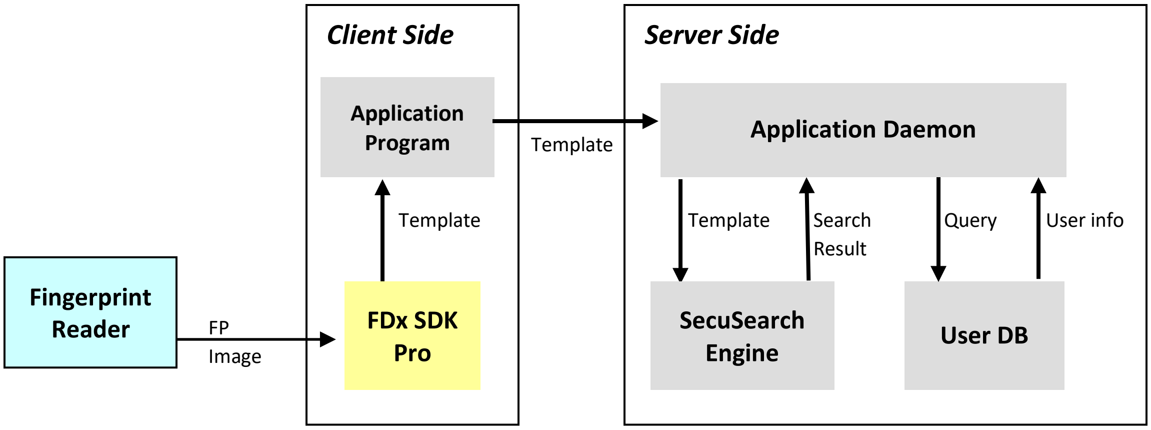
Note: In the system block diagrams below, the User DB is a database that contains registered user information. The User DB is different from the internal memory resident database managed by the SecuSearch Engine.

There are two kinds of 1:N identification systems:

Stand-Alone – This 1:N identification system has a fingerprint reader, its device driver, a fingerprint feature (minutiae) extraction module, and the SecuSearch Engine, all in one system. The application program controls each module. The following example is a ***Stand-Alone 1:N Identification System with FDx SDK Pro***:



Client-Server – This 1:N identification system has two separate systems. The client system performs fingerprint capture and fingerprint feature (minutiae) extraction functions, and it transmits the resulting fingerprint template to the server. The server transmits the search results to the client after it performs 1:N matching. The following example is a ***Client/Server 1:N Identification System with FDx SDK Pro***:

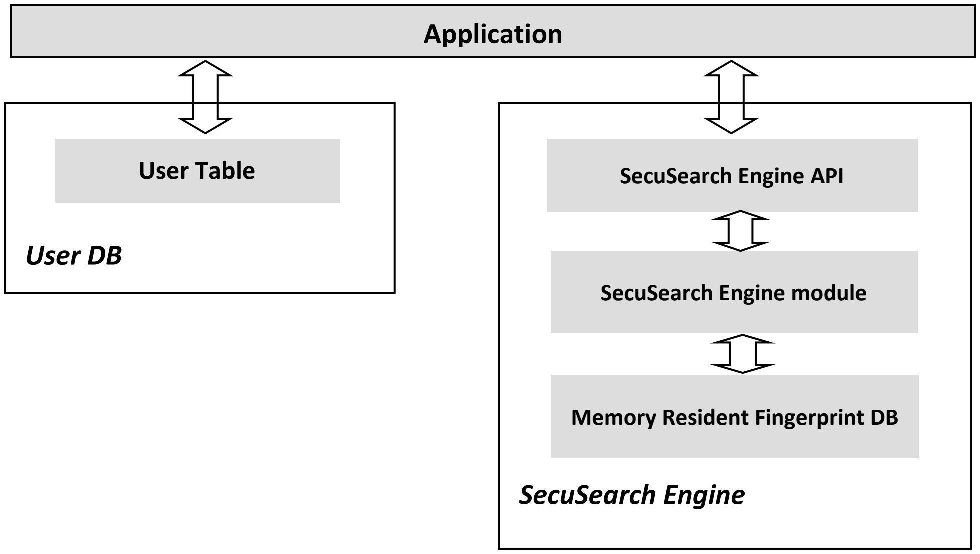


4.3. SecuSearch Database Management

The SecuSearch engine uses a highly optimized proprietary database structure to facilitate high speed fingerprint ion. It is recommended that fingerprint template data be stored in a commercial database system and then loaded into the SecuSearch database at runtime. The following databases are recommended:

- **Memory Resident Fingerprint DB**: An internal hash table-type fingerprint database that contains processed fingerprint templates and additional information used to facilitate fast searching. This memory resident fingerprint database is managed by the SecuSearch engine.
- **User DB**: A database that contains information about the registered user, the fingerprint (image) and the fingerprint template (minutiae). It is managed by the application program and should be constructed by the application programmer. It can be made using any kind of commercial database (such as MySQL, Oracle, etc.).

1:N Identification System Database Structure



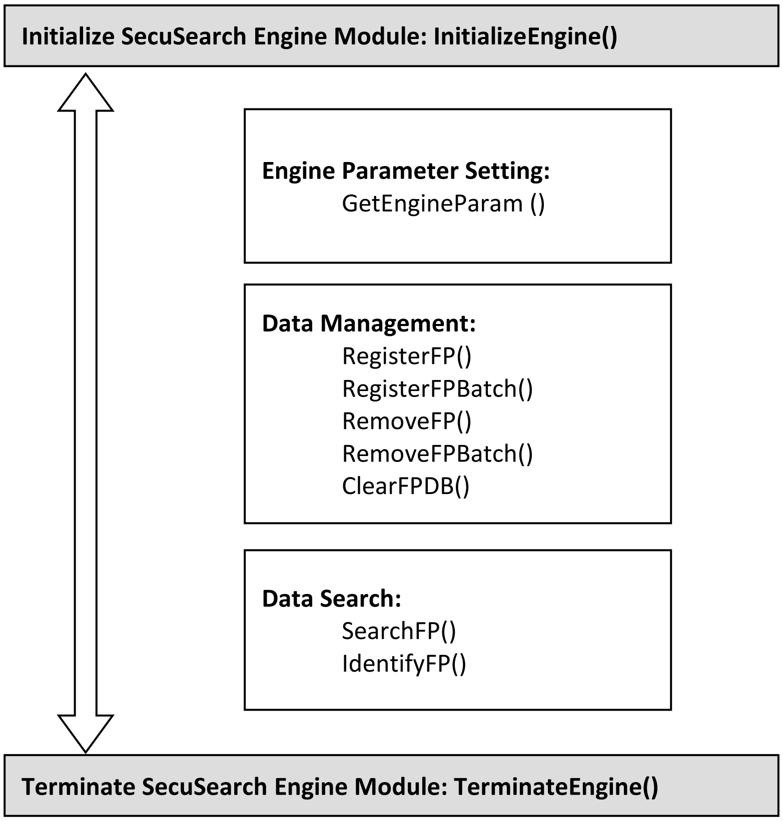
***Important**:*
*SecuSearch 3 API includes a function called **SaveFPDB()** that facilitates persisting the memory resident SecuSearch database to a disk file. This is not recommended as the primary storage method for fingerprint data. Fingerprint images and/or templates should be stored in another database or flat files so that they can be reloaded in case the SecuSearch data structure becomes corrupted.*

4.4. SecuSearchAPI C++ Class

The SecuSearch Engine APIs perform the following functions:

- Initialization and /termination
- Engine parameter setting
- Fingerprint template registration, removal, and search
- Internal fingerprint database management functions

SecuSearchAPI Structure


Initialize SecuSearch Engine Module: InitializeEngine()
Engine Parameter Setting:
GetEngineParam ()
Data Management:
RegisterFP()
RegisterFPBatch()
RemoveFP()
RemoveFPBatch()
ClearFPDB()
Data Search:
SearchFP()
IdentifyFP()
Terminate SecuSearch Engine Module: TerminateEngine()

4.4.1. SecuSearch Engine Initialization and Termination

1. Initializing SecuSearch Engine

To create a 1:N matching program using the SecuSearch Engine Module, first initialize the Engine using **InitializeEngine()**. This function initializes engine parameters, initializes the internal global variables, allocates the memory of the fingerprint database residing in the RAM, and initializes it. Be sure to use **TerminateEngine()** when exiting the host application that runs the SecuSearch Engine to free the memory allocated by the SecuSearch Engine.

When this initialization function is called, SecuSearch Engine opens the license file to check its validity. If the license file does not exist or it is not valid, initialization will fail. Since the maximum number of fingerprints that can be registered is limited by the amount licensed, users cannot register more fingerprints than the maximum number allowed.

Engine parameters at the initialization stage are defined as follows:

```
struct SSE_PARAM
{
      int32_t concurrency;
      int32_t candidateCount;
      char* szLicenseFile;
      int32_t bEnableRotation;
};
```

The **concurrency** field is used to specify the number of worker threads of SecuSearch Engine. For optimal performance, the concurrency parameter should be set to the number of logical processors on the host system. If you set the concurrency parameter as 0, SecuSearch Engine will use all the logical cores in the system automatically for the maximum speed.

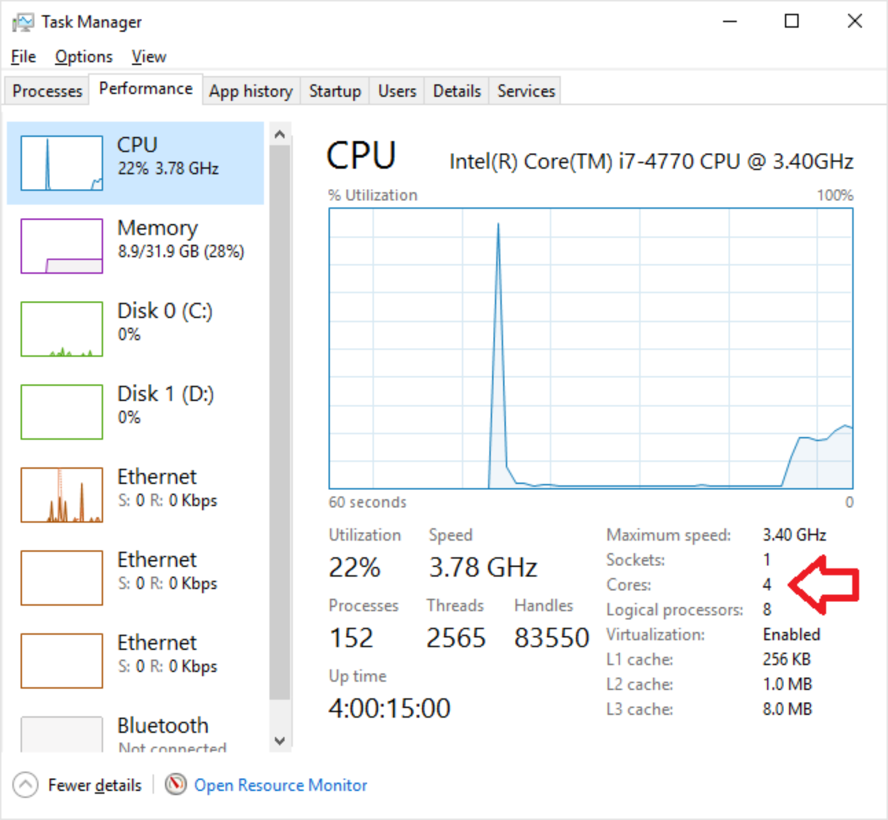
Please note that the maximum concurrency value of SecuSearch Engine 32-bit and 64-bit are 8 and 256, respectively. Therefore, the 32-bit engine can exploit up to 8 logical cores or threads regardless of the number of cores on a system.

The sample code below is for a system with four cores as shown in the following screen shot.

The **szLicenseFile** field is used to specify the location of the SecuSearch license file issued by a SecuGen representative to activate the SecuSearch Engine. The full path on the local drive should be used.

The **candidateCount** field designates the number of fingerprint candidates that will be listed when searching using the SecuSearch Engine. If, for instance, the candidateCount field is set to ‘10’, then 10 candidates will be listed. This number is typically larger in very large AFIS systems where a large number of candidates is expected. The maximum count of the candidates, `SSE_MAX_CANDIATE_COUNT`, is 300. The candidateCount is recommended to be set to no lower than ’5’.

The **bEnableRoation** field is used to specify the amount of fingerprint rotation that is allowed. If this value is set as 0, a maximally allowed rotation angle difference between a registered fingerprint and an input search fingerprint will be 45°. Otherwise, any rotation angle difference will be allowed.



CPU 22% 3.78 GHz

Memory 8.9/31.9 GB (28%)

Disk 0 (C:) 0%

Disk 1 (D:) 0%

Ethernet S: 0 R: 0 Kbps

Ethernet S: 0 R: 0 Kbps

Ethernet S: 0 R: 0 Kbps

Bluetooth Not connected

CPU Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz

Utilization 22%

Speed 3.78 GHz

Processes 152

Threads 2565

Handles 83550

Up time 4:00:15:00

Maximum speed:	3.40 GHz
Sockets:	1
Cores:	4
Logical processors:	8
Virtualization:	Enabled
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	8.0 MB

```
SSE_PARAM g_Param;    // SecuSearch Engine parameter.
SecuSearchAPI* ssapi = SecuSearch_GetAPI();
strcpy(g_Param.szLicenseFile, “c:\\SecuGen\\License.dat”);
g_Param.candidateCount = 10;

// How many threads to be exploited by the engine internally.
// If the concurrency is zero, SecuSearch Engine automatically
determines the total count of the logical CPU cores and uses all
of them.
g_Param.concurrency = 0;

// Initialize Engine
If (ssapi->InitializeEngine (&g_Param) != FPS_ERROR_NONE )
{
      // Init module failed. Show error message.
}
// Init success.
```

2. Terminating SecuSearch Engine

Close the SecuSearch Engine using **TerminateEngine()**. Be sure to use this function to free the allocated memory in the SecuSearch Engine module.

```
SSE_ERROR ret;
ret = ssapi->TerminateEngine ();//Terminate SecuSearch
```

3. Getting Engine Parameters

Use **GetEngineParam()** to get the Engine parameters after SecuSearch Engine initialization.

```
SSE_PARAM g_Param;    // SecuSearch Engine parameters
SSE_ERROR ret;
ret = sseapi.GetEngineParam (&g_Param); //Get Eng params
```

4.4.2. Fingerprint Registration and Removal

When a fingerprint is registered in SecuSearch, the fingerprint template format used is the SecuGen proprietary format (SG400), a 400-byte template that is extracted by SecuGen FDx SDK Pro.

1. Registering a Fingerprint

Use **RegisterFP()** to register a fingerprint template in the SecuSearch database. The SecuSearch Engine does not manage user information. It only takes input fingerprint data from a single fingerprint in the form of a fingerprint template. The application program must provide user management functionality by including a database (User DB) for the management of user information, such as user name, fingerprint image, and other information in addition to fingerprint template data. This is the responsibility of the application programmer, and any kind of database can be used.

```
SSE_ERROR ret;        // error code
uint8_t sg_template[SSE_TEMPLATE_SIZE]; // fingerprint template
SSE_TEMPLATE_ID template_id = 1000; // input template ID
ret = ssapi->RegisterFP( sg_template, template_id );
// register fingerprint
```

By using **RegisterFPBatch()**, you can also register a large number of fingerprint templates in a batch to greatly improve the registration speed because the internal SecuSearch Engines concurrently register the fingerprints in multiple threads.

```
SSE_ERROR ret;        // error code
ID_TEMPLATE_PAIR template_pairs[1000];
```

```
// pairs of the ID and input template

. . .
// fill the array of ID-template pairs

ret = ssapi->RegisterFPBatch( template_pairs, 1000 );
// register fingerprints in a batch
```

2. Removing a Fingerprint

Use **RemoveFP()** to delete a fingerprint template for a specific ID in the SecuSearch Engine database.

```
SSE_ERROR ret;        // error code
SSE_TEMPLATE template_id;
// a template to be removed
ret = ssapi->RemoveFP( template_id );
```

Use **RemoveFPBatch()** to delete a batch of fingerprint templates all at once in the SecuSearch Engine database.

```
SSE_ERROR ret;        // error code
SSE_TEMPLATE template_ids[10];
// an array of template IDs to be removed
ret = ssapi->RemoveFPBatch( template_ids, 10 );
```

4.4.3. Fingerprint Searching and Identification

SecuSearch supports both search and identification operations. Search operations will return a candidate list of fingerprint templates that are similar to the input (or query) fingerprint template along with corresponding confidence levels. Identification operations will search the database and identify a single fingerprint record that matches based on meeting or surpassing a pre-set security level (or threshold).

1. Searching for Similar Fingerprint Candidates

Use **SearchFP()** to get a list of fingerprint candidates that match the input (query) fingerprint. Input the fingerprint template, as extracted by the SecuGen fingerprint minutiae extraction algorithm, and the empty candidate list buffer to get the information of the closest matching fingerprint(s). The candidate list will include the template identifier (ID) and confidence level (ConfidenceLevel) of each fingerprint candidate from the database. The candidates are listed in descending order of confidence level, where a higher number means greater similarity and a lower number means less similarity to the input fingerprint. When the confidence level is higher than 5, treat the fingerprint as a match.

```
SSE_ERROR ret;                    // error code
uint8_t sg_template[SSE_TEMPLATE_SIZE];
// input fingerprint data template

CAND_LIST m_candlist;
ret = ssapi->SearchFP(sg_template, m_candlist);
// search for fingerprint in database
```

2. Identifying a Fingerprint

Use **IdentifyFP()** to identify the input (query) fingerprint template extracted by the SecuGen FDx SDK Pro based on meeting or surpassing the specified security level. If there is a matching fingerprint, the template ID will be returned in the result, otherwise the result will be set to 0.

The security level is set to determine to what degree the fingerprints match. If the resulting confidence level of the searched fingerprint is higher than the security level, the fingerprints are considered to be matching. If the confidence level is lower than the security level, the fingerprints are considered to be not matching.

```
SSE_ERROR ret;                    // error code
uint8_t sg_template[SSE_TEMPLATE_SIZE];
// input fingerprint data template
uint32_t security = CONF_LEVEL_NORMAL;      // security level
SSE_TEMPLATE_ID templateID;                 // output template ID
ret = ssapi->IdentifyFP(feat_buff, security, &templateID);
// identify the fingerprint
```

4.4.4. Managing the Fingerprint Database

The fingerprint database of the SecuSearch Engine resides in RAM and therefore must be saved to disk when a fingerprint template is registered or deleted. Also, when rebooting the system, the database stored in the disk should be loaded to memory just after initializing the SecuSearch Engine.

Important: The SaveFPDB() and LoadFPDB() storage functions in SecuSearch are not recommended as the primary storage method for fingerprint data. Fingerprint images and/or templates should be stored in another database or flat files so that they can be reloaded in case the SecuSearch data structure becomes corrupted.

1. **Saving the fingerprint database**

Use **SaveFPDB()** to save fingerprint data in the SecuSearch fingerprint database in RAM to disk

```
SSE_ERROR ret;               // error code
ret = ssapi->SaveFPDB(“sample.tdb”);
// Save SecuSearch template data to disk
```

2. **Loading the fingerprint database**

Use **LoadFPDB()**to save fingerprint data from disk into the SecuSearch fingerprint database in RAM

```
SSE_ERROR ret;               // error code
ret = ssapi->LoadFPDB(“sample.tdb”);
// Load SecuSearch template data from disk
```

3. **Clearing the fingerprint database**

Use **ClearFPDB()** to delete all fingerprint data in the SecuSearch fingerprint database in RAM

```
SSE_ERROR ret;               // error code
ret = ssapi->ClearFPDB();    // Clear all SecuSearch template data
```

4. **Getting the number of registered fingerprints in the database**

Use **GetFPCount()** to read the count of registered fingerprint templates in the memory resident fingerprint database managed by the SecuSearch Engine

```
SSE_ERROR ret;              // error code
size_t count = 0;
// Read data fingerprint count in SecuSearch database
ret = ssapi->GetFPCount(&count);
// Get the number of SecuSearch templates
```

5. Getting the list of template IDs in the database

Use **GetIDList()** to get the list of template IDs that are associated with registered fingerprint templates in the memory resident fingerprint database managed by the SecuSearch Engine

```
SSE_ERROR ret;              // error code
size_t fpCount = 0, returnedFpCount;
ret = ssapi->GetFPCount(&fpCount);

// List IDs of templates registered
SSE_TEMPLATE_ID* pIDList = new SSE_TEMPLATE_ID[fpCount];
error = ssapi->GetIDList(pIDList, fpCount, &returnedFpCount);
```

4.4.5. Converting Standard Templates into SecuGen Templates

The SecuSearch Engine only supports minutiae templates in the SecuGen proprietary format (SG400), which can contain only single finger views. Before registering and searching for a template in the ANSI 378 or ISO19794 standard format, you should extract each finger view inside the template and convert it into the SecuGen minutiae format.

1. Getting the total number of finger views in a standard template

Use **GetNumberOfView()** to get the total number of finger views in the ANSI 378 or ISO 19794 template. You can specify the second parameter of template_type into SS_TEMPLATE_ANSI378 or SS_TEMPLATE_ISO19794 according to the input template type.

```
size_t numberOfViews;

SSE_ERROR error = m_SecuSearch->GetNumberOfView(
     ansiTemplate,
     SS_TEMPLATE_ANSI378,
     &numberOfViews);
```

2. Converting a standard template into a SecuGen template

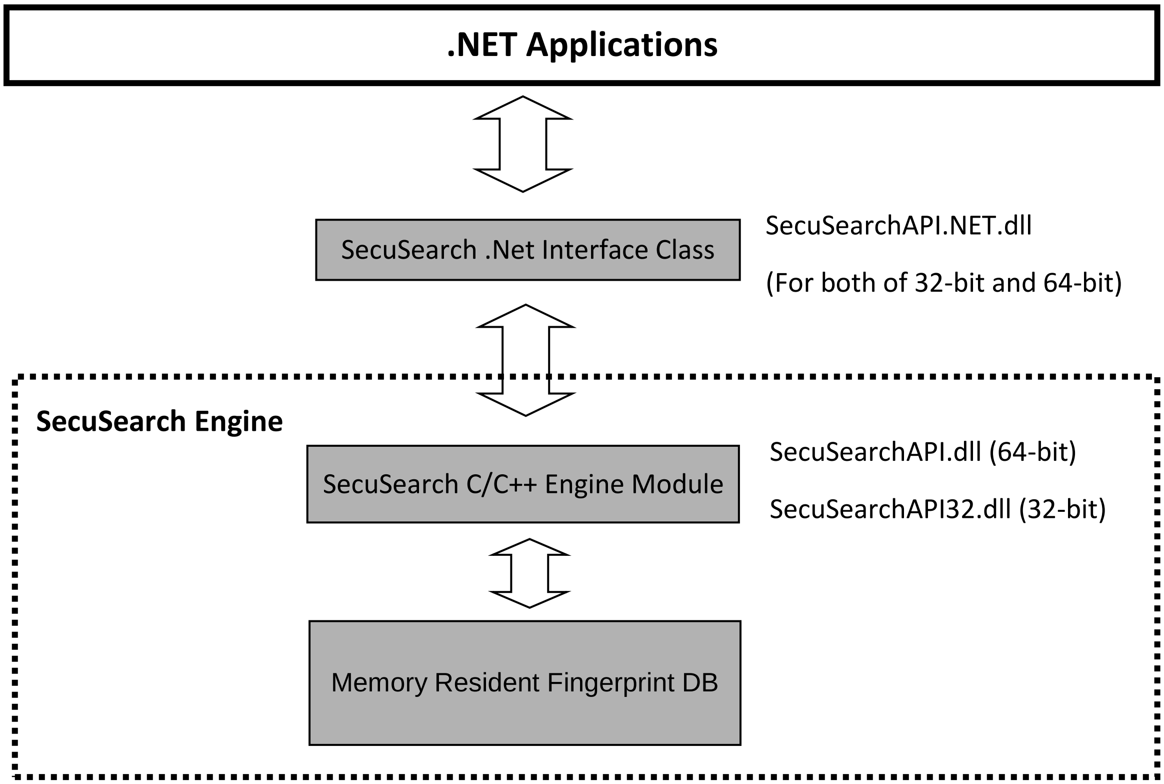
Use **ExtractTemplate()** to extract a single finger view from a standard template in the ANSI 378 or ISO 19794 format and get the finger view in the SecuGen template format. The converted template can then be used by the SecuSearch engine to register, search, or identify it.

Please note that each template should have a unique template ID to be registered.

```
SSE_ERROR ret;              // error code
uint8_t sgTemplate[SSE_TEMPLATE_SIZE];
uint32_t indexOfView;
const SSE_TEMPLATE_ID startingId = 12345;
SSE_TEMPLATE_ID templateId;

for (indexOfView = 0; indexOfView < numberOfViews; indexOfView++)
{
     error = m_SecuSearch->ExtractTemplate(
          ansiTemplate,
          SS_TEMPLATE_ANSI378,
          indexOfView,
          sgTemplate);

     templateId = startingId + indexOfView;
     error = m_SecuSearch->RegisterFP(sgTemplate, templateId);
}
```

4.5. SecuSearch .NET Class

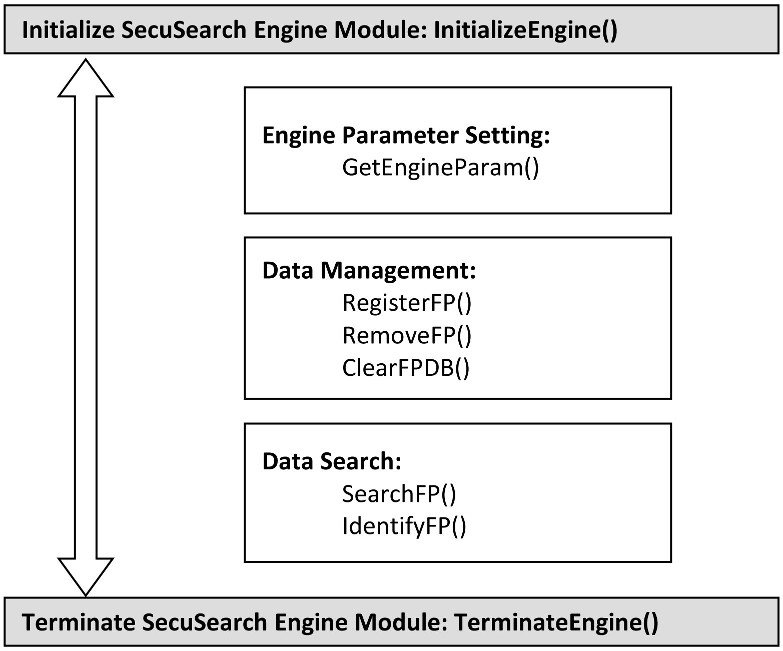
.NET applications can interface with the SecuSearch C/C++ engine by creating an instance of SecuSearch .NET class as shown in the following diagram. The SecuSearch .Net class automatically detects if an application runs on the 32-bit or 64-bit environment and internally loads a proper C/C++ DLL of SecuSearchAPI.dll or SecuSearchAPI32.dll.


.NET Applications
SecuSearch .Net Interface Class
SecuSearchAPI.NET.dll
(For both of 32-bit and 64-bit)
SecuSearch Engine
SecuSearch C/C++ Engine Module
SecuSearchAPI.dll (64-bit)
SecuSearchAPI32.dll (32-bit)
Memory Resident Fingerprint DB

The SecuSearch Engine APIs perform the following functions:

- Initialization and /termination
- Engine parameter setting
- Fingerprint registration, removal and searching
- Internal fingerprint database management functions

SecuSearch Class Structure


Initialize SecuSearch Engine Module: InitializeEngine()
Engine Parameter Setting:
GetEngineParam()
Data Management:
RegisterFP()
RemoveFP()
ClearFPDB()
Data Search:
SearchFP()
IdentifyFP()
Terminate SecuSearch Engine Module: TerminateEngine()

4.5.1. SecuSearch Engine Initialization and Termination

1. Initializing SecuSearch Engine

To create a 1:N matching program using the SecuSearch Engine Module, first initialize the Engine using **InitializeEngine()**. This function initializes engine parameters, initializes the internal global variables, allocates the memory of the fingerprint database residing in the RAM, and initializes it. Be sure to use **TerminateEngine()** when exiting the host application that runs the SecuSearch Engine to free the memory allocated by the SecuSearch Engine.

When this initialization function is called, SecuSearch Engine opens the license file to check its validity. If the license file does not exist or it is not valid, initialization will fail. Since the maximum number of fingerprints that can be registered is limited by the amount licensed, users cannot register more fingerprints than the maximum number allowed.

Engine parameters at the initialization stage are defined as follows:

```
struct SSParam
{
      int Concurrency;
      int CandidateCount;
      string LicenseFile;
      bool EnableRotation;
};
```

The **Concurrency** field is used to specify the number of worker threads of SecuSearch Engine. For optimal performance, the concurrency parameter should be set to the number of logical processors on the host system. If you set the concurrency parameter as 0, SecuSearch Engine will use all the logical cores in the system automatically for the maximum speed.

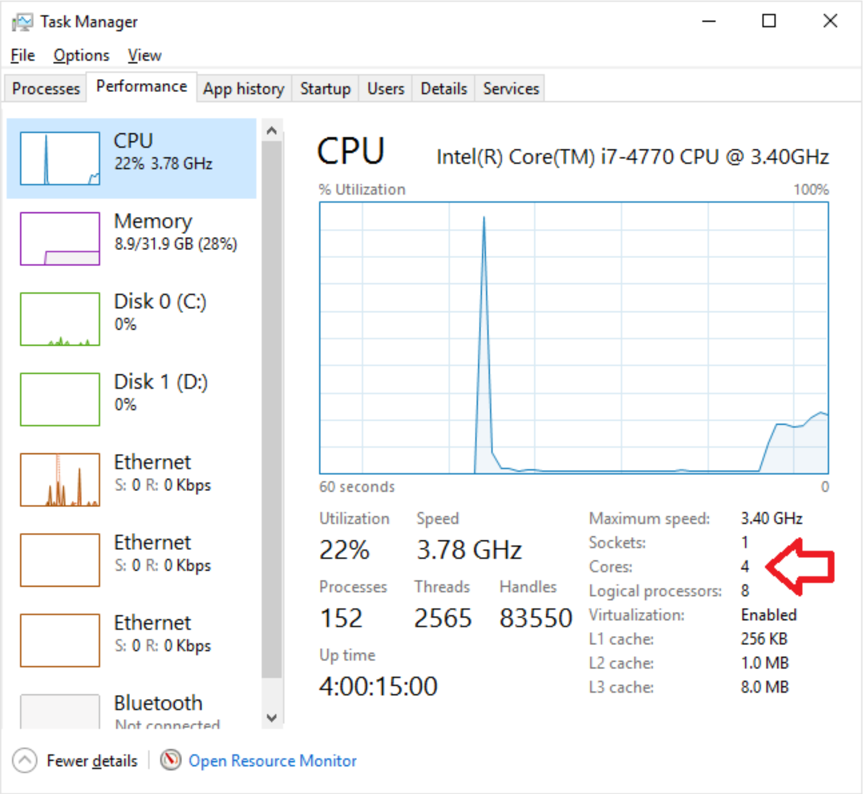
Please note that the maximum concurrency value of SecuSearch Engine 32-bit and 64-bit are 8 and 256, respectively. Therefore, the 32-bit engine can exploit up to 8 logical cores or threads regardless of the number of cores on a system.

The sample code below is for a system with four cores as shown in the following screen shot.

The **LicenseFile** field is used to specify the location of the SecuSearch license file issued by a SecuGen representative to activate the SecuSearch Engine. The full path on the local drive should be used.

The **CandidateCount** field designates the number of fingerprint candidates that will be listed when searching using the SecuSearch Engine. If, for instance, the CandidateCount field is set to '10', then 10 candidates will be listed. This number is typically larger in very large AFIS systems where a large number of candidates is expected. The maximum count of the candidates, `SSE_MAX_CANDIATE_COUNT`, is 300. The CandidateCount is recommended to be set to no lower than '5'.

The **EnableRoation** field is used to specify the limit of fingerprint rotation that is allowed. If this value is set to 0, the maximally allowed rotation angle difference between a registered fingerprint and an input search fingerprint will be 45°. Otherwise, any rotation angle difference will be allowed.



CPU Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz

Utilization 22%
Speed 3.78 GHz
Processes 152
Threads 2565
Handles 83550
Up time 4:00:15:00

Maximum speed: 3.40 GHz
Sockets: 1
Cores: 4
Logical processors: 8
Virtualization: Enabled
L1 cache: 256 KB
L2 cache: 1.0 MB
L3 cache: 8.0 MB

```
using SecuGen.SecuSearchSDK3;

SSParam param;   // SecuSearch Engine parameter.
SecuSearch SSearch = new SecuSearch();
param.LicenseFile = “.\\license.dat”;
param.CandidateCount = 10;

// How many threads to be exploited by the engine internally.
// If the concurrency is zero, SecuSearch Engine automatically
determines the total count of the logical CPU cores and uses all
of them.
param.Concurrency = 0;

// Whether or not to allow any amount of fingerprint rotations.
param.EnableRotation = true;

// Initialize Engine
if (SSearch.InitializeEngine(param) != SSError.NONE )
```

```
{
     // Init module failed. Show error message.
}
// Init success.
```

2. Terminating SecuSearch Engine

Close the SecuSearch Engine using **TerminateEngine()**. Be sure to use this function to free the allocated memory in the SecuSearch Engine module.

```
SSError error; // error code
error = SSearch.TerminateEngine();//Terminate SecuSearch
```

3. Getting Engine Parameters

Use **GetEngineParam()** to get the Engine parameters after SecuSearch Engine initialization.

```
SSParam param;   // SecuSearch Engine parameters
SSError error;  // error code
error = SSearch.GetEngineParam(ref param); //Get Eng params
```

4.5.2. Fingerprint Registration and Removal

When a fingerprint is registered in SecuSearch, the fingerprint template format used is the SecuGen proprietary format (SG400), a 400-byte template that is extracted by SecuGen FDx SDK Pro.

1. Registering a Fingerprint

Use **RegisterFP()** to register a fingerprint template in the SecuSearch database. The SecuSearch Engine does not manage user information. It only takes input fingerprint data from a single fingerprint in the form of a fingerprint template. The application program must provide user management functionality by including a database (User DB) for the management of user information, such as user name, fingerprint image, and other information in addition to fingerprint template data. This is the responsibility of the application programmer, and any kind of database can be used.

```
SSError error;   // error code
byte[] sgTemplate; // fingerprint template
uint templateId = 1000; // input template ID
error = SSearch.RegisterFP(sgTemplate, templateId);
// register fingerprint
```

By using **RegisterFPBatch()**, you can also register a large number of fingerprint templates in a batch to greatly improve the registration speed because the internal SecuSearch Engines concurrently register the fingerprints in multiple threads.

```
SSError error;        // error code
SSIdTemplatePair[] pairs = new SSIdTemplatePair[1000];
// pairs of the ID and input template

. . .
// fill the array of ID-template pairs

error = SSearch.RegisterFPBatch(pairs, 1000);
// register fingerprints in a batch
```

2. Removing a Fingerprint

Use **RemoveFP()** to delete a fingerprint template for a specific ID in the SecuSearch Engine database.

```
SSError error;        // error message
uint template_id;
// a template to be removed
```

```
error = SSearch.RemoveFP(template_id);
```

4.5.3. Fingerprint Searching and Identification

SecuSearch supports both search and identification operations. Search operations will return a candidate list of fingerprint templates that are similar to the input (or query) fingerprint template along with corresponding confidence levels. Identification operations will search the database and identify a single fingerprint record that matches based on meeting or surpassing a pre-set security level (or threshold).

1. Searching for Similar Fingerprint Candidates

Use **SearchFP()** to get a list of fingerprint candidates that match the input (query) fingerprint. Input the fingerprint template, as extracted by the SecuGen fingerprint minutiae extraction algorithm, and the empty candidate list buffer to get the information of the closest matching fingerprint(s). The candidate list will include the template identifier (ID) and confidence level (ConfidenceLevel) of each fingerprint candidate from the database. The candidates are listed in descending order of confidence level, where a higher number means greater similarity and a lower number means less similarity to the input fingerprint. When the confidence level is higher than 5, treat the fingerprint as a match.

```
byte[] sgTemplate;    // input fingerprint data template

// create or load a template into sgTemplate …

SSError error;        // error code
SSCandList candlist = new SSCandList();
error = SSearch.SearchFP(sgTemplate, ref candlist);
// search for fingerprint in database
```

2. Identifying a Fingerprint

Use **IdentifyFP()** to identify the input (query) fingerprint template extracted by the SecuGen FDx SDK Pro based on meeting or surpassing the specified security level. If there is a matching fingerprint, the template ID will be returned in the result, otherwise the result will be set to 0.

The security level is set to determine to what degree the fingerprints match. If the resulting confidence level of the searched fingerprint is higher than the security level, the fingerprints are considered to be matching. If the confidence level is lower than the security level, the fingerprints are considered to be not matching.

```
byte[] sgTemplate;    // input fingerprint data template
```

```
// create or load a template into sgTemplate …

SSError error;        // error code
SSConfLevel security = SSConfLevel.NORMAL;  // security level
uint templateID;                 // output template ID
error = SSearch.IdentifyFP(sgTemplate, security, ref templateID);
// identify the fingerprint
```

4.5.4. Managing the Fingerprint Database

The fingerprint database of the SecuSearch Engine resides in RAM and therefore must be saved to disk when a fingerprint template is registered or deleted. Also, when rebooting the system, the database stored in the disk should be loaded to memory just after initializing the SecuSearch Engine.

Important: The SaveFPDB() and LoadFPDB() storage functions in SecuSearch are not recommended as the primary storage method for fingerprint data. Fingerprint images and/or templates should be stored in another database or flat files so that they can be reloaded in case the SecuSearch data structure becomes corrupted.

1. Saving the fingerprint database

Use **SaveFPDB()** to save fingerprint data in the SecuSearch fingerprint database in RAM to disk

```
SSError error;             // error code
error = SSearch.SaveFPDB(“sample.tdb”);
// Save SecuSearch template data to disk
```

2. Loading the fingerprint database

Use **LoadFPDB()**to save fingerprint data from disk into the SecuSearch fingerprint database in RAM

```
SSError error;             // error code
error = SSearch.LoadFPDB(“sample.tdb”);
// Load SecuSearch template data from disk
```

3. Clearing the fingerprint database

Use **ClearFPDB()** to delete all fingerprint data in the SecuSearch fingerprint database in RAM

```
SSError error;             // error code
error = SSearch.ClearFPDB();// Clear all SecuSearch template data
```

4. Getting the number of registered fingerprints in the database

Use **GetFPCount()** to read the count of registered fingerprint templates in the memory resident fingerprint database managed by the SecuSearch Engine

```
SSError error;              // error code
int count = 0;
// Read data fingerprint count in SecuSearch database
error = SSearch.GetFPCount(ref count);
// Get the number of SecuSearch templates
```

5. Getting the list of template IDs in the database

Use **GetIDList()** to get the list of template IDs that are associated with registered fingerprint templates in the memory resident fingerprint database managed by the SecuSearch Engine

```
SSError error;              // error code
// List IDs of templates registered
List<uint> idList = new List<uint>();

error = SSearch.GetIDList(idList);
```

4.5.5. Converting Standard Templates into SecuGen Templates

The SecuSearch Engine only supports minutiae templates in the SecuGen proprietary format (SG400), which can contain only single finger views. Before registering and searching for a fingerprint template in the ANSI 378 or ISO19794 standard format, you should extract each finger view inside the template and convert it into the SecuGen minutiae format.

1. Getting the total number of finger views in a standard template

Use **GetNumberOfView()** to get the total number of finger views in the ANSI 378 or ISO 19794 template. You can specify the second parameter of templateType into SSTemplateType.ANSI378 or SSTemplateType.ISO19794 according to the input template type.

```
SSError error;              // error code
uint numberOfViews;

error = SSearch.GetNumberOfView(
     ansiTemplate,
     SSTemplateType.ANSI378,
     ref numberOfViews);
```

2. Converting a standard template into a SecuGen template

Use **ExtractTemplate()** to extract a single finger view from a standard template in the ANSI 378 or ISO 19794 format and get the finger view in the SecuGen template format. The converted template can then be used by the SecuSearch engine to register, search, or identify it.

Please note that each template should have a unique template ID to be registered.

```
SSError error;        // error code
byte[] sgTemplate = new byte[SSConstants.TEMPLATE_SIZE];
uint indexOfView;
uint startingId = 123450;
uint templateId;

for (indexOfView = 0; indexOfView < numberOfViews; indexOfView++)
{
      error = SSearch.ExtractTemplate(
            ansiTemplate,
            SSTemplateType.ANSI378,
            indexOfView,
            sgTemplate);

      templateId = startingId + indexOfView;
      error = SSearch.RegisterFP(sgTemplate, templateId);
}
```

Appendix A. API Reference

Structures, error codes, and declared values are defined in ‘SecuSearchDef.h.’ The C and C++ API functions are defined in ‘SecuSearchAPI.h’ and ‘SecuSearchAPI.hpp’, respectively.

A.1. Functions

A.1.1. SecuSearch Engine Initialization, Termination and Configuration

C	**SSE_ERROR SecuSearch_InitializeEngine(const SSE_PARAM* param);**
C++	**SSE_ERROR SecuSearchAPI::InitializeEngine(const SSE_PARAM* param);**
.NET	**SSError SecuSearch.InitializeEngine(SSParam param);**

Inputs user setting parameters of SecuSearch Engine to initialize SecuSearch Engine. This function checks the validity of the license file, allocates memory for the internal in-memory fingerprint database, and initializes global variables. Refer to SSE_PARAM structure for more information.

- **Parameters**

  param: [IN] Pointer to SSE_PARAM structure containing search engine parameters

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Failed to initialize

  FPS_ERROR_LOW_MEM: Failed to allocate memory

C	**SSE_ERROR SSE_API SecuSearch_TerminateEngine(void);**
C++	**SSE_ERROR SecuSearchAPI::TerminateEngine(void);**
.NET	**SSError SecuSearch.TerminateEngine();**

Terminates SecuSearch Engine. When exiting the 1:N matching system, this function should be called. This function frees all the memory that the SecuSearch Engine has allocated.

- **Parameters**

  None

- **Return values**

  FPS_ERROR_NONE: Error None

  FPS_ERROR_NOT_INIT: Not initialized

C	**SSE_ERROR SecuSearch_GetEngineParam(SSE_PARAM* param);**
C++	**SSE_ERROR SecuSearchAPI::GetEngineParam(SSE_PARAM* param);**
.NET	**SSError SecuSearch.GetEngineParam(ref SSParam param);**

Returns the parameter values of the current SecuSearch Engine configuration,

- **Parameters**

  param: [OUT] Pointer to SSE_PARAM structure that will return search engine parameters

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

A.1.2. Fingerprint Registration and Removal

C	**SSE_ERROR SecuSearch_RegisterFP(const uint8_t *sg_template, SSE_TEMPLATE_ID template_id);**
C++	**SSE_ERROR SecuSearchAPI::RegisterFP(const uint8_t* sg_template, SSE_TEMPLATE_ID template_id);**
.NET	**SSError SecuSearch.RegisterFP(byte[] sgTemplate, uint templateId);**

Registers fingerprint template. Inputs 400-byte fingerprint minutiae template extracted by FDx SDK Pro and template ID to register a fingerprint to the internal memory resident fingerprint database of the SecuSearch Engine.

- **Parameters**

  Sg_template: [IN] Pointer to byte array that contains a 400-byte fingerprint template

  template_id: [IN] SSE_TEMPLATE_ID variable that contains an unsigned integer template id

- **Return values**

  FPS_ERROR_NONE: No error

  FPS_ERROR_NOT_INIT: Not initialized

  FPS_ERROR_OVER_LIMIT: Exceeded the maximum number of fingerprints to register

  FPS_ERROR_DATA_EXIST: The template ID is already registered

  FPS_ERROR_INVALD_TEMPLATE: Invalid template

  FPS_ERROR_TOO_FEW_MINUTIAE: The template has too few minutiae

  FPS_ERROR_TOO_FEW_FEAT: The template does not have enough features for processing

  FPS_ERROR_LOW_MEM: Failed memory allocation

C	**SSE_ERROR SecuSearch_RegisterFPBatch(const ID_TEMPLATE_PAIR *pairs, size_t count);**
C++	**SSE_ERROR SecuSearchAPI::RegisterFPBatch(const ID_TEMPLATE_PAIR* pairs, size_t count);**
.NET	**SSError SecuSearch.RegisterFPBatch(SSIDTemplatePair[] pairs, int count);**

Registers fingerprint templates in a batch. All the internal SecuSearch engines concurrently register the given templates so that the registration speed can much faster than the speed of calling RegisterFP multiple times, especially in cases with a large batch count.

- **Parameters**

  pairs: [IN] Pointer to ID_TEMPLATE_PAIR array that contains a SSE_TEMPLATE_ID and a 400-byte fingerprint template

  count: [IN] size_t variable that contains the total count of the pairs

- **Return values**

  FPS_ERROR_NONE: No error

  FPS_ERROR_NOT_INIT: Not initialized

  FPS_ERROR_OVER_LIMIT: Exceeded the maximum number of fingerprints to register

FPS_ERROR_DATA_EXIST: The template ID is already registered

FPS_ERROR_INVALD_TEMPLATE: Invalid template

FPS_ERROR_TOO_FEW_MINUTIAE: The template has too few minutiae

FPS_ERROR_TOO_FEW_FEAT: The template does not have enough features for processing

FPS_ERROR_LOW_MEM: Failed memory allocation

C	**SSE_ERROR SecuSearch_RemoveFP(SSE_TEMPLATE_ID template_id);**
C++	**SSE_ERROR SecuSearchAPI::RemoveFP(SSE_TEMPLATE_ID template_id);**
.NET	**SSError SecuSearch.RemoveFP(uint templateId);**

Removes a template for the specified ID from the database.

- **Parameters**

  template_id: [IN] SSE_TEMPLATE_ID variable that contains an unsigned integer template id

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

  FPS_ERROR_DATA_NOT_FOUND: There is no template with the given ID

C	**SSE_ERROR SecuSearch_RemoveFPBatch(SSE_TEMPLATE_ID *template_ids, size_t count);**
C++	**SSE_ERROR SecuSearchAPI::RemoveFP(SSE_TEMPLATE_ID *template_ids, size_t count);**
.NET	**SSError SecuSearch.RemoveFP(uint[] templateId, int count);**

Removes templates for the specified IDs from the database.

- **Parameters**

  template_ids: [IN] Pointer to SSE_TEMPLATE_ID array that contains unsigned integer template IDs

  count: [IN] size_t variable that contains the total count of the template IDs

- **Return values**

FPS_ERROR_NONE: Error none

FPS_ERROR_NOT_INIT: Not initialized

A.1.3. Fingerprint Searching and Identification

C	**SSE_ERROR SecuSearch_SearchFP(const uint8_t *sg_template, CAND_LIST* candList);**
C++	**SSE_ERROR SecuSearchAPI::SearchFP(const uint8_t* sg_template, CAND_LIST& candList);**
.NET	**SSError SecuSearch. SearchFP(byte[] sgTemplate, ref SSCandList candList);**

Searches for matching fingerprint candidates in the SecuSearch database that are similar to the input 400-byte template extracted by FDx SDK Pro. This function returns a list of similar fingerprint candidates. The candidates are listed in descending order of confidence level that quantifies the degree of similarity between the compared fingerprints. If the confidence level is above 5, it can be regarded that the two fingerprints match. The number of candidates in the list can be set by calling SecuSearchAPI::SetParameter() .

- **Parameters**

  sg_template: [IN] Pointer to a byte array that contains a 400-byte fingerprint template

  candList: [OUT] CAND_LIST structure to receive similar fingerprint candidates

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

  FPS_ERROR_INVALD_TEMPLATE: Invalid template

  FPS_ERROR_TOO_FEW_MINUTIAE: The template has too few minutiae

  FPS_ERROR_TOO_FEW_FEAT: The template does not have enough features for processing

C	SSE_ERROR SecuSearch_IdentifyFP(const uint8_t *sg_template, uint32_t seculevel, SSE_TEMPLATE_ID *template_id);
C++	SSE_ERROR SecuSearchAPI::IdentifyFP(const uint32_t* sg_template, uint32_t seculevel, SSE_TEMPLATE_ID *template_id);
.NET	SSError SecuSearch.IdentifyFP(byte[] sgTemplate, SSConfLevel seculevel, ref uint templateId);

Identifies an input (query) fingerprint. This function determines whether there is a fingerprint in the SecuSearch memory resident database that matches the input fingerprint. A 400-byte fingerprint minutiae template extracted by FDx SDK Pro and a security level are passed as inputs. If a matching fingerprint is found, this function returns FPS_ERROR_NONE and the identified template ID. Otherwise, this function returns FPS_ERROR_ IDENTIFICATION_FAIL. The security level is a threshold that is used to decide whether two fingerprints match. The security level can be set to a value from 1 to 9. If the security level is set to high, identification will succeed only when there is a high degree of correlation between the minutiae points of the two fingerprints.

- **Parameters**

  sg_template: [IN] Pointer to a byte array that contains a 400-byte fingerprint template

  seculevel: [IN] A security level (or threshold) that must be one of the following:

CONF_LEVEL_LOWEST	1
CONF_LEVEL_LOWER	2
CONF_LEVEL_LOW	3
CONF_LEVEL_BELOW_NORMAL	4
CONF_LEVEL_NORMAL	5
CONF_LEVEL_ABOVE_NORMAL	6
CONF_LEVEL_HIGH	7
CONF_LEVEL_HIGHER	8
CONF_LEVEL_HIGHEST	9

  template_id: [OUT] The template ID of the matching template or INVALID_SSE_TEMPLATE_ID if no match was found

- **Return values**

  FPS_ERROR_NOT_INIT: Not initialized

  FPS_ERROR_NONE: Input fingerprint is identified

  FPS_ERROR_IDENTIFICATION_FAIL: Input fingerprint is not identified

  FPS_ERROR_INVALD_TEMPLATE: Invalid template

  FPS_ERROR_TOO_FEW_MINUTIAE: The template has too few minutiae

  FPS_ERROR_TOO_FEW_FEAT: The template does not have enough features for processing

A.1.4. Database Management

C	**SSE_ERROR SecuSearch_SaveFPDB(const char* filename);**
C++	**SSE_ERROR SecuSearchAPI::SaveFPDB(const char* filename);**
.NET	**SSError SecuSearch.SaveFPDB(string filename);**

Saves all fingerprint data in the internal memory resident fingerprint database of SecuSearch Engine to a disk file

- **Parameters**

  filename: [IN] String containing a file path

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

***Important**: SaveFPDB() and LoadFPDB() are not recommended as the primary storage method for fingerprint data. Fingerprint images and/or templates should be stored in another database or flat files so that they can be reloaded in the event that the SecuSearch data structure becomes corrupted.*

C	**SSE_ERROR SecuSearch_LoadFPDB(const char* filename);**
C++	**SSE_ERROR SecuSearchAPI::LoadFPDB(const char* filename);**
.NET	**SSError SecuSearch.LoadFPDB(string filename);**

Loads fingerprint data into the internal memory resident fingerprint database of SecuSearch Engine from a disk files

- **Parameters**

  filename: [IN] String containing a file path

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

C	**SSE_ERROR SecuSearch_ClearFPDB();**
C++	**SSE_ERROR SecuSearchAPI::ClearFPDB();**
.NET	**SSError SecuSearch.ClearFPDB();**

Removes all fingerprint data in the internal memory resident fingerprint database of SecuSearch Engine

- **Parameters**

  None

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

C	**SSE_ERROR SecuSearch_GetTemplate(SSE_TEMPLATE_ID template_id, uint8_t* sg_template);**
C++	**SSE_ERROR SecuSearchAPI:: GetTemplate(SSE_TEMPLATE_ID template_id, uint8_t* sg_template);**
.NET	**SSError SecuSearch.GetTemplate(uint templateId, byte[] sgTemplate);**

Gets a registered template with a given ID

- **Parameters**

  template_id: [IN] A template ID to get

  sg_template: [OUT] Pointer to a byte array that will contain a 400-byte fingerprint template

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

  FPS_ERROR_DATA_NOT_FOUND: There is no template with the given ID

C	**SSE_ERROR SecuSearch_GetFPCount(size_t* count);**
C++	**SSE_ERROR SecuSearchAPI::GetFPCount(size_t* count);**
.NET	**SSError SecuSearch.GetFPCount(ref int count);**

Reads the count of registered fingerprints in the memory resident fingerprint database managed by SecuSearch Engine

- **Parameters**

  count: [OUT] Pointer to size_t variable that will contain the count of fingerprint templates in the database

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

C	**SSE_ERROR SecuSearch_GetIDList(SSE_TEMPLATE_ID* pIDList, size_t maxCount, size_t* count);**
C++	**SSE_ERROR SecuSearchAPI::GetIDList(SSE_TEMPLATE_ID* pIDList, size_t maxCount, size_t* count);**
.NET	**SSError SecuSearch.GetIDList(List<uint> idList, Int32 maxCount);**

Reads the list of template IDs of registered fingerprints in the memory resident database of the SecuSearch Engine

- **Parameters**

  pIDList: [OUT] Pointer to SSE_TEMPLATE_ID array that will contain the IDs associated with fingerprint templates in the database

  maxCount: [IN] Parameter specifying the size of the pIDList array

  count: [OUT] the count of the returned IDs in the pIDList array

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

A.1.5. Standard template conversion

C	**SSE_ERROR SecuSearch_GetNumberOfView(const uint8_t* standard_template, uint32_t template_type, size_t *number_of_view);**
C++	**SSE_ERROR SecuSearchAPI::GetNumberOfView(const uint8_t* standard_template, uint32_t template_type, size_t *number_of_view);**
.NET	**SSError SecuSearch.GetNumberOfView(byte[] standardTemplate, SSTemplateType templateType, ref uint numberOfView);**

Gets the total number of finger views in the ANSI 378 or ISO 19794 template.

- **Parameters**

  standard_template: [IN] Pointer to a byte array that contains a standard template in the ANSI 378 or ISO 19794 format

  template_type: [IN] Unsigned integer that represents the template format of the given *standard_template*. It can be *SS_TEMPLATE_ANSI387* or *SS_TEMPLATE_ISO19794*.

  number_of_view: [OUT] Pointer to a size_t variable to receive the total number of finger views in the given *standard_template.*

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

  FPS_ERROR_INVALD_TEMPLATE: Invalid standard_template

  FPS_ERROR_INVALID_PARAM: Invalid template_type parameter

C	**SSE_ERROR SecuSearch_ExtractTemplate(const uint8_t* standard_template,** **uint32_t template_type,** **uint32_t index_of_view,** **uint8_t* sg_template);**
C++	**SSE_ERROR SecuSearchAPI::ExtractTemplate(const uint8_t* standard_template,** **uint32_t template_type,** **uin32_t index_of_view,** **uint32_t* sg_template);**
.NET	**SSError SecuSearch.ExtractTemplate(byte[] standardTemplate,** **SSTemplateType templateType,** **uint numberOfView,** **Byte[] sgTemplate);**

Extracts a single finger view from a standard template in the ANSI 378 or ISO 19794 format, and converts it into a template in the SecuGen template format (SG400). The returned *sg_template* buffer can be used by the SecuSearch engine to register, search, or identify it.

- **Parameters**

  standard_template: [IN] Pointer to a byte array that contains a standard template in the ANSI 378 or ISO 19794 format

  template_type: [IN] Unsigned integer that represents the template format of the given *standard_template*. It can be *SS_TEMPLATE_ANSI387* or *SS_TEMPLATE_ISO19794*.

  index_of_view: [IN] Unsigned integer that specifies a finger view to extract as a zero-based index inside the given *standard_template*

  sg_template: [OUT] Pointer to a byte array that will contain a template of the specified finger view in the SecuGen proprietary template format. The buffer size should not be less than *SSE_TEMPLATE_SIZE*.

- **Return values**

  FPS_ERROR_NONE: Error none

  FPS_ERROR_NOT_INIT: Not initialized

  FPS_ERROR_INVALD_TEMPLATE: Invalid standard_template

  FPS_ERROR_INVALID_PARAM: Invalid template_type or index_of_view parameter

A.1.6. Version String

C/C++	**const char* SecuSearch_GetVersion(void);**
.NET	**string SecuSearch.GetVersion();**

Gets the version string in the format: “Major.Minor.Revision”. For example, it returns “3.1.2” when the major version is 3, the minor version is 1, and the revision number is 2.

- **Return values**

  The version string in the format: “Major.Minor.Revision”

A.2. SecuSearch Engine Data Structures

C/C++

```
struct SSE_PARAM
{
      int concurrency;
      int candidateCount;
      char* szLicenseFile;
      int bEnableRoation;
};
```

.NET

```
struct SSParam

{
      int Concurrency;
      int CandidateCount;
      string LicenseFile;
      bool EnableRoation;
};
```

Description: SecuSearch Engine parameters set by user

Members:

concurrency: Number of CPU cores for SecuSearch Engine to use. If the value is set to zero, SecuSearch Engine will use all the CPU cores on a system for the maximum speed.

candidateCount: Number of fingerprint candidates resulting from a search

szLicenseFile: Full path of SecuSearch license file

bEnableRotation: Specifies the allowed amount of fingerprint rotation angle difference between a registered fingerprint and a search fingerprint. If it is zero, the maximally allowed rotation angle difference will be 45°. Otherwise, any rotation angle difference will be allowed.

C/C++	**typedef uint32_t SSE_TEMPLATE_ID;**
.NET	**uint**

Description: Unique identifier of a fingerprint template

C/C++

```
struct CANDIDATE
{
      SSE_TEMPLATE_ID id;
      int matchScore;
      int confidenceLevel;
};
```

.NET

```
struct SSCandidate
{
      uint id;
      int MatchScore;
      int ConfidenceLevel;
};
```

Description: User fingerprint information

Members:

id: Template ID number in user database

matchScore: Matching score generated by SecuGen MINEX algorithm

confidenceLevel: Confidence level (similarity between input and result fingerprint) (1-9)

C/C++

```
struct CAND_LIST
{
      struct CANDIDATE candidates[SSE_MAX_CANDIDATE_COUNT];
      int count;
};
```

.NET

```
struct SSCandList
{
      SSCandidate[] Candidates;
      int Count;
};
```

Description: Fingerprint candidate list as a search result

Members:

candidates: Structure containing details of candidates found

count: The number of candidates found

C/C++

struct ID_TEMPLATE_PAIR

```
{
      SSE_TEMPLATE_ID id;
      uint8_t template[SSE_TEMPLATE_SIZE];
};
```

.NET

struct SSIdTemplatePair

```
{
      uint id;
      byte[] template;
};
```

Description: A pair of a template ID and 400-byte fingerprint template associated with the ID

Members:

id: Template ID number in user database

template: 400-byte fingerprint template associated with this ID

A.3. Error Codes & Defined Constants

A.3.1. Error Codes

Error Code	Value	Definition
FPS_ERROR_NONE	0	No error
FPS_ERROR_NOT_INIT	1	Engine is not initialized
FPS_ERROR_INVALID_PARAM	2	Invalid parameter
FPS_ERROR_SAVE_DB	102	Failed to save database
FPS_ERROR_LOAD_DB	103	Failed to load database
FPS_ERROR_INVALD_TEMPLATE	104	Invalid Template
FPS_ERROR_DATA_EXIST	105	Data already exists
FPS_ERROR_OVER_LIMIT	201	Exceeded the maximum number of fingerprints allowed to be registered (per license)
FPS_ERROR_IDENTIFICATION_FAIL	202	Input fingerprint is not identified
FPS_ERROR_TOO_FEW_MINUTIAE	301	The template has too few minutiae
FPS_ERROR_TOO_FEW_FEAT	302	The template does not have enough features for processing
FPS_ERROR_LICENSE_LOAD	501	Failed to load license file
FPS_ERROR_LICENSE_KEY	502	Invalid license key
FPS_ERROR_LICENSE_EXPIRED	503	License is expired
FPS_ERROR_LICENSE_WRITE	504	Failed to create license file
FPS_ERROR_HEAP_ALLOC	2004	Memory error
FPS_ERROR_HEAP_FREE	2005	Memory error
FPS_ERROR_SET_LOCK_PAGE_PRIVILEGE	2006	No privilege for Lock Pages in Memory. Refer to the Distribution Guide for more information
FPS_ERROR_ALLOC_PHYS_MEM	2007	Memory error
FPS_ERROR_FREE_PHYS_MEM	2008	Memory error
FPS_ERROR_MAP_PHYS_MEM	2009	Memory error
FPS_ERROR_RESERVE_MEM	2010	Memory error
FPS_ERROR_LOW_MEM	2011	Memory error
FPS_ERROR_TOO_MANY_FEAT	2101	Memory error
FPS_ERROR_DATA_NOT_FOUND	2103	No data found

A.3.2. Defined Constants

CONF_LEVEL_LOWEST	1
CONF_LEVEL_LOWER	2
CONF_LEVEL_LOW	3
CONF_LEVEL_BELOW_NORMAL	4
CONF_LEVEL_NORMAL	5
CONF_LEVEL_ABOVE_NORMAL	6

CONF_LEVEL_HIGH	7
CONF_LEVEL_HIGHER	8
CONF_LEVEL_HIGHEST	9
FIN_NUM_UNKNOWN	0
FIN_NUM_RIGHT_THUMB	1
FIN_NUM_RIGHT_INDEX	2
FIN_NUM_RIGHT_MIDDLE	3
FIN_NUM_RIGHT_RING	4
FIN_NUM_RIGHT_LITTLE	5
FIN_NUM_LEFT_LITTLE	6
FIN_NUM_LEFT_INDEX	7
FIN_NUM_LEFT_MIDDLE	8
FIN_NUM_LEFT_RING	9
FIN_NUM_LEFT_LITTLE	10
SS_TEMPLATE_SG400	0
SS_TEMPLATE_FIR	1
SS_TEMPLATE_ANSI378	2
SS_TEMPLATE_ISO19794	3
SG_TEMPLATE_SIZE	400
SSE_TEMPLATE_SIZE	400
INVALID_SSE_TEMPLATE_ID	0xFFFFFFFF
SSE_MAX_CANDIDATE_COUNT	300

Appendix B. 64-bit Engine Performance

B.1. Benchmark Environment

All benchmark results have been measured on a system having the following specifications.

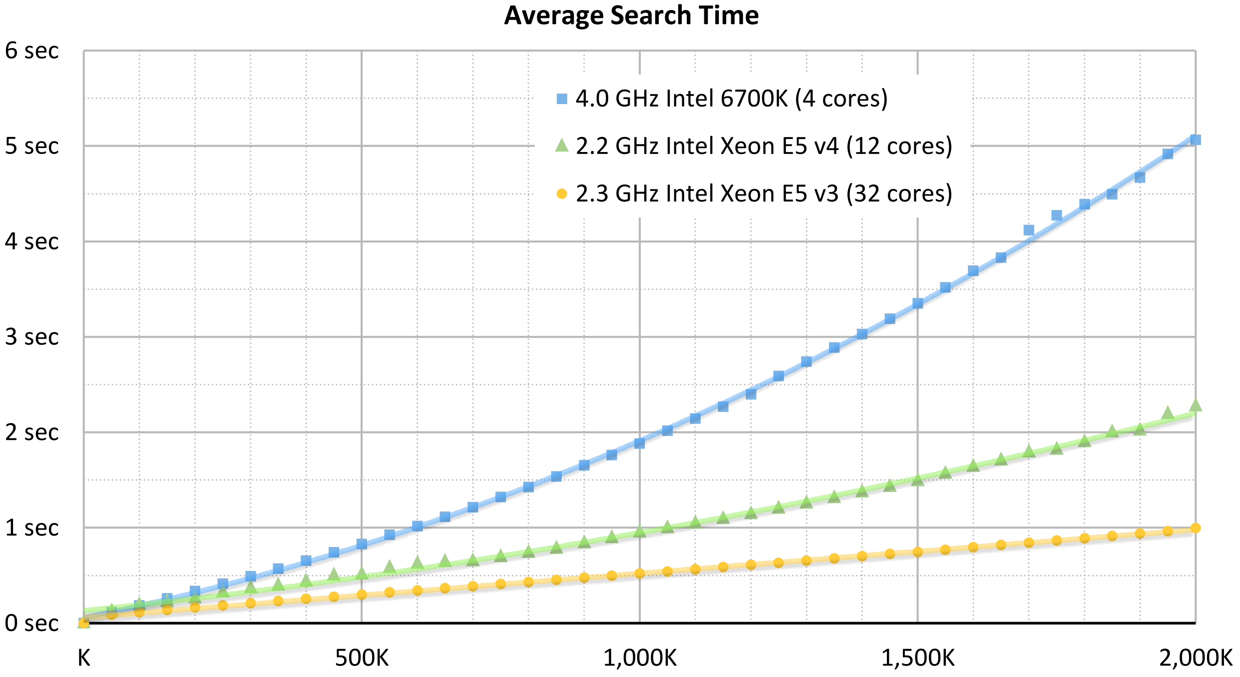
CPU	RAM	O/S
4.0GHz Intel® Core™ i7 6700K (4 cores, 8 hyperthreads)	32GB	Windows 10 Pro
2.2GHz Intel® Xeon™ E5 v4 (12 cores, 24 hyperthreads)	32GB	Windows Server 2016
2.3GHz Intel® Xeon™ E5 v3 (32 cores, 64 hyperthreads)	32GB	Windows Server 2016

All the templates used for the benchmark are extracted from fingerprint images of which size is 300x400 in pixels. Please note that the following processing times would be reduced if templates from smaller images were used because smaller images usually have fewer minutiae and the processing time is roughly proportional to the minutiae count of a template.

B.2. Search Time

The following figure shows the average time to search templates with the 10-candidate setting at each count of registered template. For example, the average search time on the Intel 6700K CPU is about 1 second when the total registered template count is 600,000.

In addition, the search time is decreased in proportion to the increase of the number of CPU cores.



Average Search Time

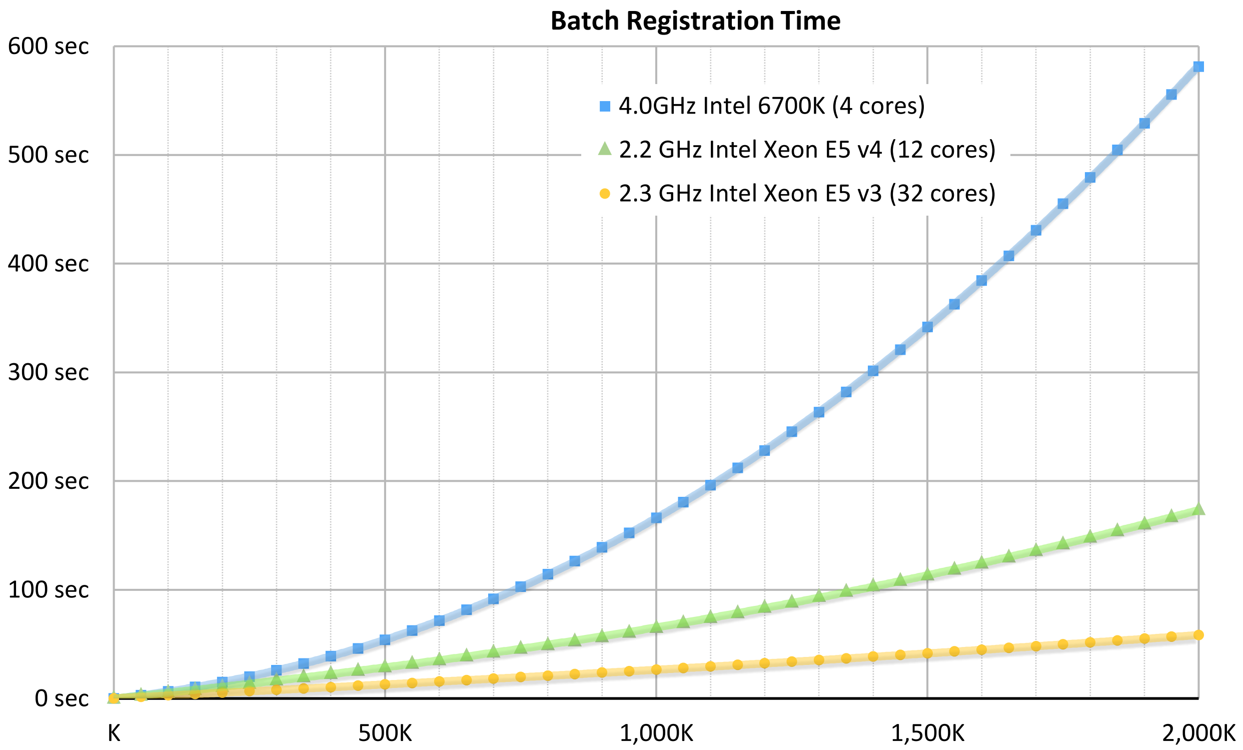
Templates	4.0GHz Intel 6700K (4 cores)	2.2GHz Xeon E5 v4 (12 cores)	2.3GHz Intel Xeon E5 v3 (32 cores)
50K	0.113 sec	0.123 sec	0.087 sec
100K	0.184 sec	0.192 sec	0.114 sec
200K	0.335 sec	0.262 sec	0.164 sec
300K	0.489 sec	0.369 sec	0.209 sec
400K	0.654 sec	0.439 sec	0.254 sec
500K	0.829 sec	0.514 sec	0.298 sec
600K	1.018 sec	0.625 sec	0.342 sec
700K	1.215 sec	0.652 sec	0.387 sec
800K	1.424 sec	0.739 sec	0.431 sec
900K	1.652 sec	0.838 sec	0.475 sec
1,000K	1.883 sec	0.952 sec	0.520 sec
1,100K	2.142 sec	1.045 sec	0.564 sec
1,200K	2.398 sec	1.146 sec	0.610 sec
1,300K	2.740 sec	1.255 sec	0.656 sec
1,400K	3.029 sec	1.372 sec	0.701 sec
1,500K	3.349 sec	1.488 sec	0.746 sec
1,600K	3.692 sec	1.642 sec	0.794 sec
1,700K	4.118 sec	1.790 sec	0.842 sec
1,800K	4.388 sec	1.899 sec	0.889 sec
1,900K	4.669 sec	2.021 sec	0.938 sec
2,000K	5.065 sec	2.273 sec	0.997 sec

B.3. Registration Time

B.3.1. Batch Registration Time

The following figure shows the total time of the batch registration by using `RegisterFPBatch` at various template counts, where the template count in a batch is 1,000. For example, the time to register 200,000 templates is 15 seconds.

In addition, the batch registration time is decreased in proportion to the increase of the number of CPU cores.



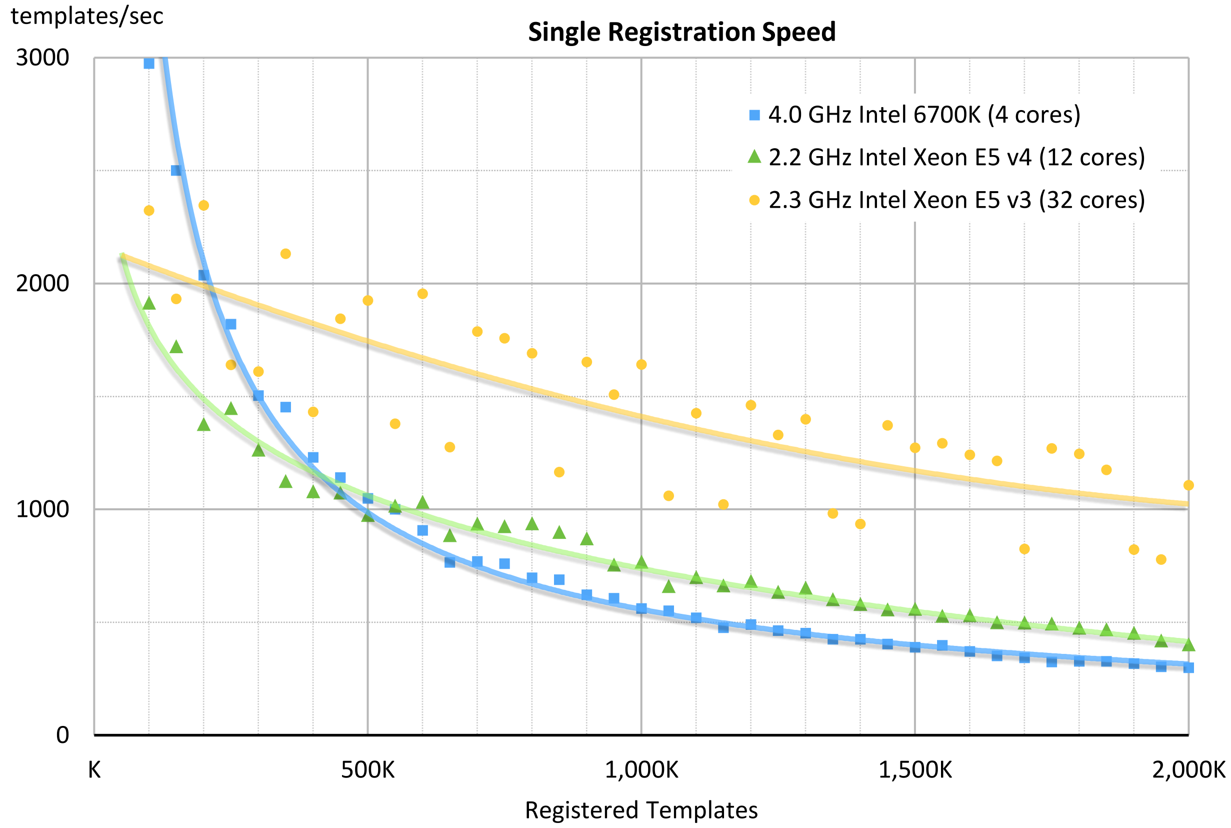
Batch Registration Time

Templates	4.0GHz Intel 6700K (4 cores)	2.2GHz Xeon E5 v4 (12 cores)	2.3GHz Intel Xeon E5 v3 (32 cores)
50K	3.0 sec	2.5 sec	1.5 sec
100K	6.5 sec	5.2 sec	2.8 sec
200K	14.9 sec	10.5 sec	5.3 sec
300K	25.8 sec	16.6 sec	7.9 sec
400K	38.7 sec	22.5 sec	10.3 sec
500K	53.7 sec	28.7 sec	12.8 sec
600K	71.4 sec	35.4 sec	15.5 sec
700K	91.6 sec	42.4 sec	18.2 sec
800K	114.1 sec	49.2 sec	20.9 sec
900K	139.0 sec	56.6 sec	23.6 sec
1,000K	166.1 sec	65.0 sec	26.5 sec
1,100K	196.2 sec	74.1 sec	29.3 sec
1,200K	228.1 sec	83.6 sec	32.3 sec
1,300K	263.3 sec	93.5 sec	35.3 sec
1,400K	301.1 sec	103.2 sec	38.4 sec
1,500K	341.5 sec	113.3 sec	41.5 sec
1,600K	384.2 sec	124.2 sec	44.7 sec
1,700K	430.7 sec	135.7 sec	48.0 sec
1,800K	479.1 sec	148.0 sec	51.4 sec
1,900K	529.1 sec	160.2 sec	54.9 sec
2,000K	581.2 sec	173.5 sec	58.4 sec

B.3.2. Single Registration Speed

The following figure shows the registration speed of `RegisterFP` at various counts of registered templates. For example, the SecuSearch engine on Intel 6700K CPU can handle 1,001 requests of the single template registration per second by using `RegisterFP`, where the total count of the registered template is 500,000.

Please note that the registration speed of a single template is not increased in proportion to the number of CPU cores as much in the batch registration because the `RegisterFP` always exploits a single core.



Single Registration Speed (templates/sec)

Templates	4.0GHz Intel 6700K (4 cores)	2.2GHz Xeon E5 v4 (12 cores)	2.3GHz Intel Xeon E5 v3 (32 cores)
50K	2,975	1,909	2,324
100K	2,501	1,715	1,933
200K	1,821	1,442	1,641
300K	1,453	1,119	2,132
400K	1,142	1,068	1,845
500K	1,001	1,009	1,380
600K	766	879	1,276
700K	760	919	1,758
800K	689	894	1,166
900K	607	749	1,508
1,000K	551	654	1,060
1,100K	477	657	1,022
1,200K	463	628	1,330
1,300K	426	595	982
1,400K	404	550	1,372
1,500K	399	522	1,293
1,600K	351	494	1,215
1,700K	325	488	1,270
1,800K	328	462	1,175
1,900K	304	413	779
2,000K	301	386	993

Appendix C. 32-bit Engine Performance

C.1. Benchmark System

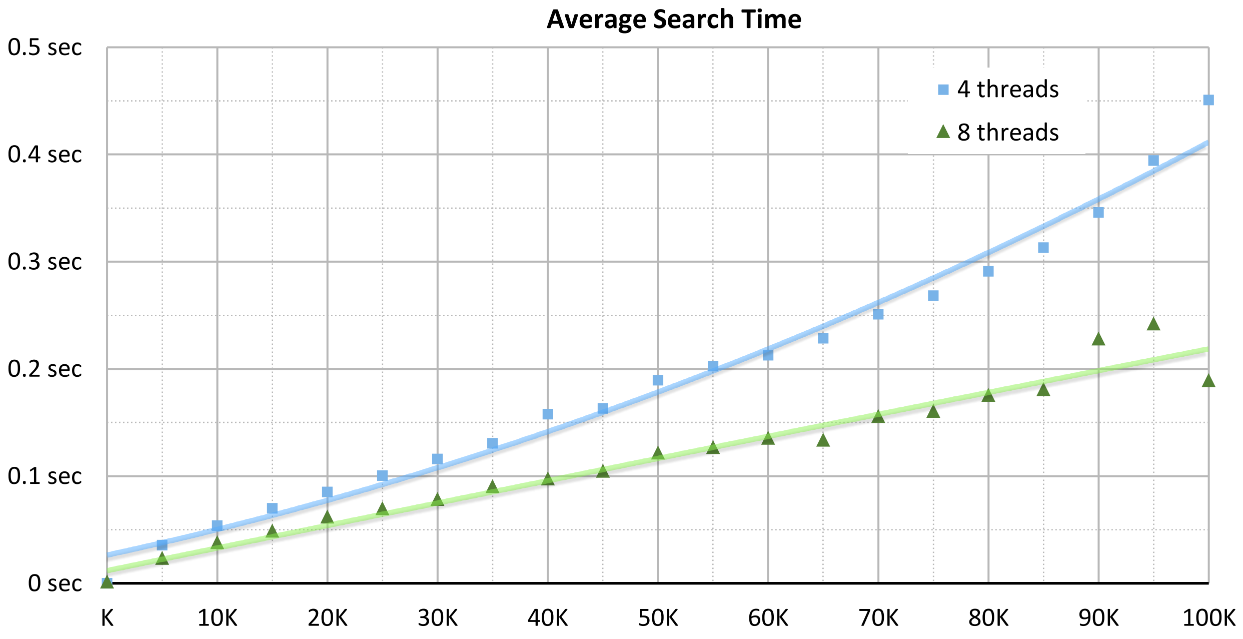
All benchmark results have been measured on a system having the following specifications.

CPU	RAM	O/S
2.2GHz Intel® Xeon™ E5 v4 (12 cores, 24 hyperthreads)	32GB	Windows Server 2016

C.2. Search Time

The following figure shows the average time to search templates with the 10-candidate setting at each count of registered template. For example, the average search time is about 0.1 second when the total registered template count is 40,000 and the engine uses 8 threads.

The search time can be decreased in proportion to the increase of the number of CPU cores or threads. In addition, the maximum concurrency value or thread count for the 32-bit engine is 8 so that it can exploit up to 8 logical or physical CPUs even if a system has more CPU cores.



Average Search Time

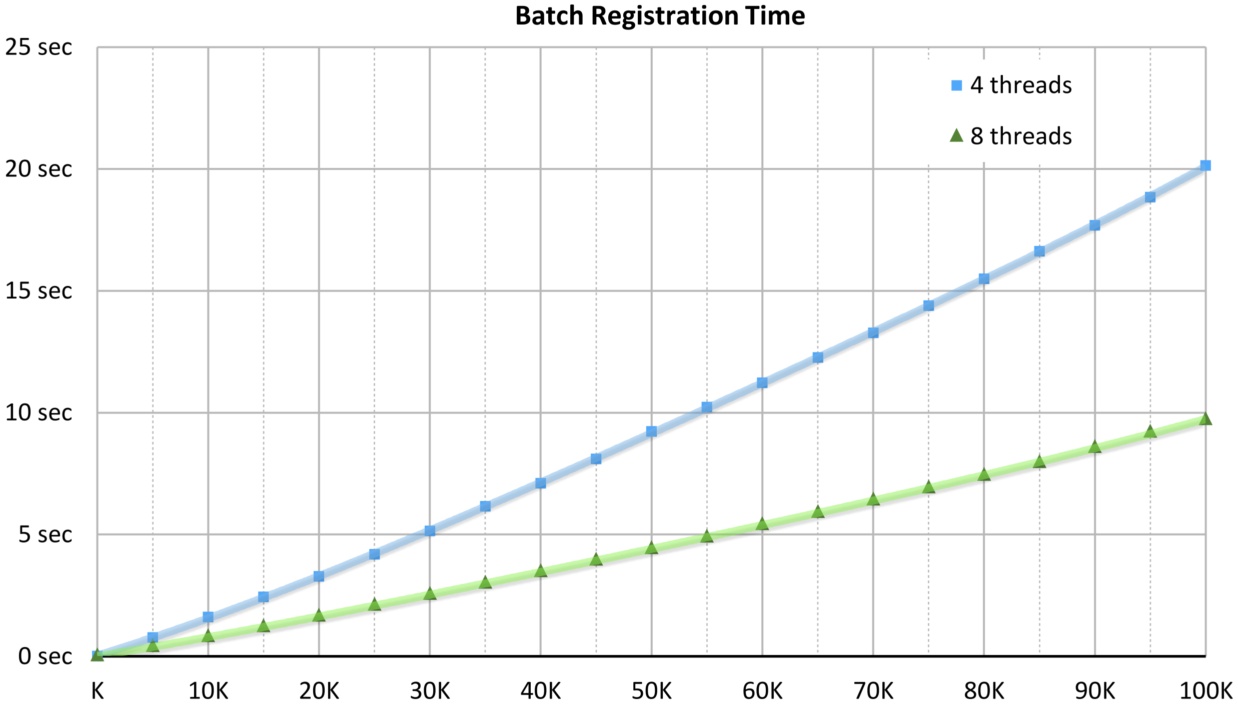
Templates	2.3GHz Xeon E5 v4	
	4 threads	8 threads
5K	0.036 sec	0.022 sec
10K	0.054 sec	0.037 sec
15K	0.070 sec	0.047 sec
20K	0.085 sec	0.061 sec
25K	0.100 sec	0.068 sec
30K	0.116 sec	0.077 sec
35K	0.131 sec	0.089 sec
40K	0.158 sec	0.096 sec
45K	0.163 sec	0.103 sec
50K	0.189 sec	0.121 sec
55K	0.203 sec	0.125 sec
60K	0.213 sec	0.134 sec
65K	0.228 sec	0.132 sec
70K	0.251 sec	0.155 sec
75K	0.268 sec	0.159 sec
80K	0.291 sec	0.174 sec
85K	0.313 sec	0.180 sec
90K	0.346 sec	0.227 sec
95K	0.394 sec	0.241 sec
100K	0.451 sec	0.188 sec

C.3. Registration Time

C.3.1. Batch Registration Time

The following figure shows the total time of the batch registration by using `RegisterFPBatch` at various template counts, where the template count in a batch is 1,000. For example, the time to register 100,000 templates by using 8 threads is about 10 seconds.

In addition, the batch registration time is decreased in proportion to the increase of the number of CPU cores or threads.



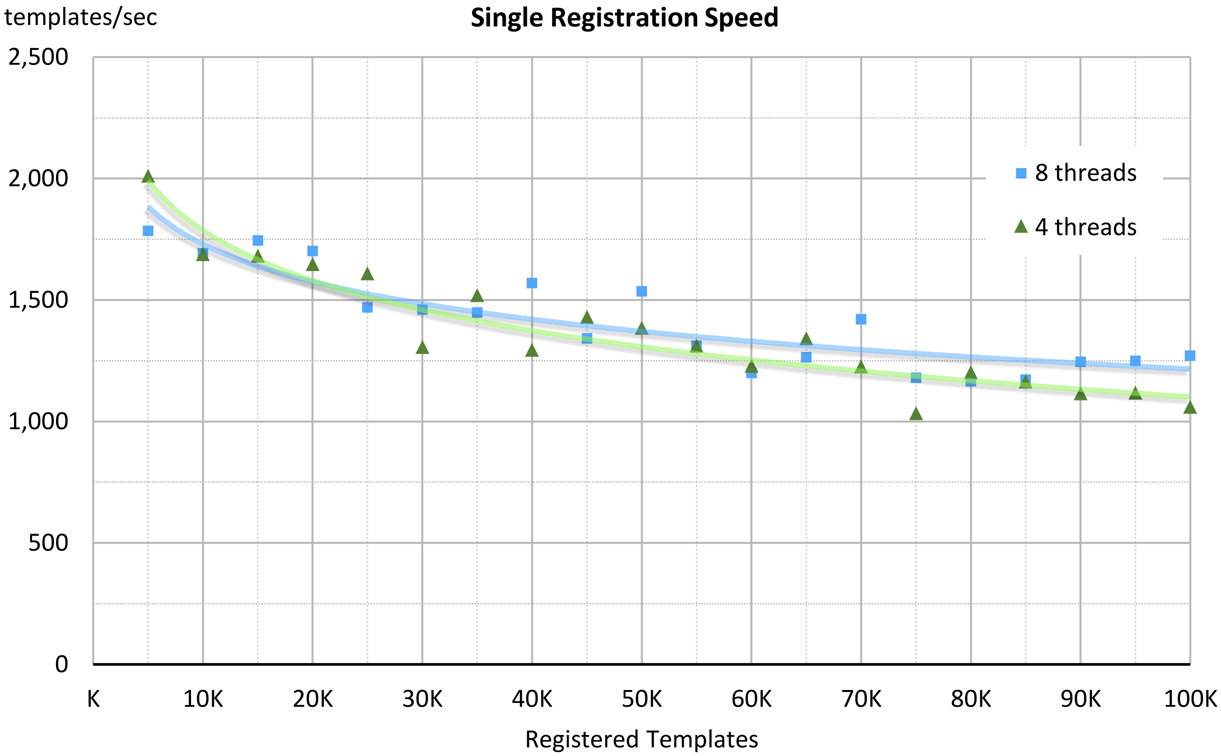
Batch Registration Time

Templates	2.3GHz Xeon E5 v4	
	4 threads	8 threads
5K	0.8 sec	0.4 sec
10K	1.6 sec	0.8 sec
15K	2.4 sec	1.2 sec
20K	3.3 sec	1.6 sec
25K	4.2 sec	2.1 sec
30K	5.1 sec	2.5 sec
35K	6.2 sec	3.0 sec
40K	7.1 sec	3.5 sec
45K	8.1 sec	3.9 sec
50K	9.2 sec	4.4 sec
55K	10.2 sec	4.9 sec
60K	11.2 sec	5.4 sec
65K	12.3 sec	5.9 sec
70K	13.3 sec	6.4 sec
75K	14.4 sec	6.9 sec
80K	15.5 sec	7.4 sec
85K	16.6 sec	7.9 sec
90K	17.7 sec	8.5 sec
95K	18.8 sec	9.2 sec
100K	20.1 sec	9.7 sec

C.3.2. Single Registration Speed

The following figure shows the registration speed of RegisterFP at various counts of registered templates. For example, the SecuSearch engine can handle 1,272 requests of the single template registration per second by using RegisterFP, where the total count of the registered template is 100,000 and the engine exploits 8 threads.

Please note that the registration speed of a single template is not increased in proportion to the number of CPU cores or threads because the RegisterFP always exploits a single core.



Single Registration Speed (templates/sec)

Templates	2.2GHz Xeon E5 v4	
	4 threads	8 threads
5K	2,005	1,785
10K	1,683	1,693
15K	1,676	1,746
20K	1,641	1,702
25K	1,603	1,471
30K	1,300	1,461
35K	1,514	1,449
40K	1,288	1,571
45K	1,424	1,343
50K	1,380	1,536
55K	1,306	1,312
60K	1,223	1,201
65K	1,336	1,265
70K	1,220	1,421
75K	1,028	1,180
80K	1,197	1,166
85K	1,158	1,173
90K	1,109	1,246
95K	1,112	1,251
100K	1,054	1,272

Appendix D. About SecuGen Fingerprint Readers

For more information about SecuGen fingerprint readers and sensors, please visit our website at: www.secugen.com/products

SecuGen Sensor Qualities

- **Excellent Image Quality**: Clear, distortion-free fingerprint images are generated using advanced, patented 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 Fingerprint Sensors

- Unique optical method captures fine details, even from dry skin
- Extremely low image-distortion for greater accuracy
- Resistance to damaging electrostatic discharge, moisture or corrosion
- Superior mechanical strength, wear-resistance and durability with no need for costly coatings
- Broad range of applicability, especially for use in extreme conditions and climates
- Low cost, long life, and no maintenance requirements – suitable for mass deployments