NIST ELFT-EFS
Evaluation of Latent Fingerprint Technology — Extended Feature Sets
Evaluation 2
Test Plan / API / Schedule

12 April 2010

Contents

1 Overview

The NIST Evaluation of Latent Fingerprint Technology — Extended Feature Sets (ELFT-EFS) is an independently administered technology evaluation of latent fingerprint feature-based matching systems. ELFT-EFS is being conducted by the National Institute of Standards & Technology (NIST).

ELFT-EFS is a complement to NIST's Evaluation of Latent Fingerprint Technology (ELFT) testing program. The ELFT evaluations to date have focused solely on automated feature extraction and matching (AFEM) in the context of latent fingerprint identification.

ELFT-EFS will evaluate the accuracy of latent matching using features marked by experienced human latent fingerprint examiners. The purpose of this test is to evaluate the current state of the art in latent feature-based matching, by comparing the accuracy of searches using images alone with searches using different feature sets. The features sets will include the current IAFIS latent feature set, and different subsets of the Extended Feature Set (EFS) features proposed by CDEFFS[1]. A key result of the test is to determine when human feature markup is effective. Because human markup is expensive in terms of time, effort, and expertise, there is a need to know when image-only searching is adequate, and when the additional effort of marking minutiae and extended features is appropriate.

The following summarizes the planned test:

- The evaluation will involve 1:N searches using latent 1000ppi images provided with human markup of EFS features.
- Exemplars for the gallery will be images only. Exemplars will be 500ppi.
- The test will be an SDK-type test, in that participants will provide software, and all processing will take place on NIST hardware.
- Different tests will be run for the following search types:
  - Image only
  - Image with region of interest markup
  - Image with minutiae (IAFIS EFTS LFFS equivalent)
  - Image with EFS features
  - Minutiae only (IAFIS EFTS LFFS equivalent)

Test results will be made publicly available in a NIST report after the conclusion of the test.

2 Participation

Participation in Evaluation 2 is open to all developers of latent fingerprint identification systems.

All systems must comply with the API outlined in Section 5.4. Anonymous participation will not be permitted. The Application form[2] includes details regarding application and qualification.

3 Data

3.1 Datasets

Validation Dataset

A Validation Dataset will be provided to participants before the evaluation to verify the correct operation of participants' software before and after delivery to NIST.

Evaluation Dataset

The latent and exemplar images and features in Evaluation #2 will be similar but not identical to those in Evaluation #1. The Evaluation Dataset will contain sequestered data, formatted in the same manner as the Validation Dataset. The Evaluation Dataset will contain Privacy Act or FOIA Protected Information and will not be released to the participants or the public. The Evaluation Dataset will to the extent permitted by law be protected under the Freedom of Information Act (5 U.S.C 552) and the Privacy Act (5 U.S.C. 552a) as applicable.

[1] *CDEFFS is the ANSI/NIST Committee to Define an Extended Fingerprint Feature Set. The current working draft of the Extended Fingerprint and Palmprint Features document can be found at http://fingerprint.nist.gov/standard/cdeffs/.*

[2] *The Application form can be found at http://fingerprint.nist.gov/latent/elft-efs/*

3.2 Format

All images and data will be contained in ANSI/NIST files. All images will be 8-bit grayscale.

Each latent ANSI/NIST file in the evaluation will contain one Type-1 record, one Type-2 record, zero or one Type-9 records, and one Type-13 record. All latent images will be in Type-13 records, in uncompressed format.

Each exemplar ANSI/NIST file in the evaluation will contain one Type-1 record, and ten Type-14 records (one for each finger, with finger positions identified). All exemplar images will be in Type-14 records. 500ppi exemplar images will be compressed using WSQ.

3.3 Features

Files containing exemplars will not have any features defined: no Type-9 record will be present.

Files containing latents may or may not have any features defined: zero or one Type-9 records will be present. There will be tests comparing the accuracy of two primary types of searches:

- Image-only searches, in which the latent image will not be accompanied by a type-9 record.
- Feature-based searches, in which the latent image will be accompanied by a type-9 record with features defined in fields 9.300-9.372, formatted in accordance with "Data Format for the Interchange of Extended Fingerprint and Palmprint Features," abbreviated here as the "EFS Spec" (Extended Feature Set Specification). The test will evaluate different combinations of EFS fields, so not all EFS fields may be present in any given search. The subsets of features used (defined as Subsets LA-LG) are defined in Section 7.

Note: The current EFS Spec version is 0.4 (June 2009).

All of the latent IAFIS/EFS features will be provided with feature markup by human experts. Note that all human markup will be conducted outside of ELFT-EFS and is not part of the evaluation.

Note also that conformance testing of automatic extraction of CDEFFS features is not part of this test. In other words, the evaluation will not be measuring how close automatically extracted features are to examiner created features. Automated algorithms can use the extended features defined for a latent search without explicitly computing them for the exemplar image, and thus it must be emphasized that automated extraction of the extended features on the exemplar is not necessarily the only nor the best way to use this information. For example, an examiner may mark an area as a scar; for the exemplar, the matcher would not necessarily have to mark the area as a scar, but may use that information to match against a corresponding area with many false minutiae and poor ridge flow.

3.4 Resolution

All latent images will be 1000 pixels per inch.

Exemplar images will be at 500 pixels per inch. This resolution will be contained in field 14.009 (Horizontal pixel scale), which will be identical to field 14.010 (Vertical pixel scale).

3.5 Dimensions and orientation

Latent fingerprint images may vary from 0.3"x 0.3" to 2.0" x 2.0" (width x height), all at 1000ppi. 1st & 3rd quartiles are about 700-1200 pixels (width) or 900-1400 pixels (height).

Exemplar images will be approximately upright (in the same orientation as they were captured).

Neither latent nor exemplar images will be larger than 2.0" in either width or height.

When orientation is known in advance or discernable by an examiner, latent fingerprint images will not vary in orientation from upright more than ±45°. However, there are images in the Evaluation Dataset for which orientation cannot be determined in advance, and all possible orientations (±180°) must be considered equally likely. Images from latent subtests LB-LG will

include the orientation direction and uncertainty fields (9.301). Images from latent subtest LA will not.

3.6 Exemplar types

All exemplars will include rolled or plain (segmented slap) fingerprints. The impression types will include optical livescan and inked paper sources. The impression type will be noted in field 14.003.

Exemplars will always include all ten fingers, and are therefore referred to here as a 10-finger exemplar set (also commonly called a ten print set).

Note that a 10-finger exemplar set will consist of either ten rolled prints, or ten plain prints.

In some cases, multiple sets of 10-finger exemplar sets associated with one person will be included in the gallery. This association will be made explicit in the exemplar enrollment stage: at the time of enrollment, exemplars that are known to belong to the same person will always share the same subject ID.

3.7 Finger positions

Exemplars will be provided in complete 10-finger sets, all contained within a single ANSI/NIST file, with finger positions noted.

The finger positions for latents will not be noted – no searches will be restricted to specific fingers.

3.8 Dataset size

The largest size gallery used for Evaluation 2 will contain 100,000 subjects having two 10-finger exemplar sets (rolled and plain impressions) per subject.

The total number of unique latent images is approximately 1,100, with the number of latent searches based on section 4.2.

4 Evaluation Criteria

4.1 Performance Metrics

Performance metrics will be based on rank and matcher score:

- Rank will be reported by the number of true matches reported in each position in the candidate list. For example, the Rank-1 metric is the proportion of searches in which the correct mate appears in the top position on the candidate list. CMC[3] curves will also be reported to show how many latent images are correctly identified at rank 1, rank 2, etc. A CMC is a plot of identification rate vs. recognition rank. Identification rate at rank k is the proportion of the latent images correctly identified at rank K or lower. A latent image has rank k if its mate is the kth largest comparison score on the candidate list. Recognition rank ranges from 1 to 100, as 100 is the (maximum) candidate list size specified in the API.

- Matcher score metrics are evaluated in terms of DET/ROC[4] performance, by plotting False Positive Identification Rate (FPIR) and False Negative Identification Rate (FNIR) for all score values. Note that this approach requires that a given matcher score be comparable between different latent searches. Both the absolute matcher score and the probability of true match values (see Section 5.6) will be used for DET analysis.

[3] *Cumulative Match Characteristic*
[4] *Detection Error Trade-off/Receiver Operating Characteristic*

4.2 Evaluation Subtests

The Evaluation is composed of the following subtests. For precise definitions of which features will be present for each subtest: see Section 7. All latents in each subtest may or may not be searched against all exemplars (galleries). Participants have the option of specifying which specific subtest combinations they wish to be tested on.

- Latent Subtests
  - LA – image only
  - LB – image + ROI
  - LC – image + ROI + Pattern Class + Quality Map
  - LD – image + IAFIS/EFTS equivalent features
  - LE – image + baseline EFS
  - LF – image + baseline EFS + Skeleton
  - LG – IAFIS/EFTS equivalent features only
- Exemplar Subtests
  - E1 – 100,000 subjects; 1 set of 10 rolled and 1 set of 10 plain impressions each; 500ppi

4.3 Reporting of Results

The ELFT-EFS Final Report will contain descriptive information concerning the evaluation, descriptions of each experiment, aggregate test results across all participants, and individual test results for each participant. All results will be reported for each participating system, with the exception of results for different combinations of EFS features. Because not all participating systems may implement all of the EFS features, results from those evaluations will be stated in generic terms so that participants cannot deduce which features are used by other systems.

Note that the application form stipulates that each participant consents to the disclosure of its performance.

Enrollment, feature extraction and search timing information will also be reported, with the explicit caveat that speed of execution, for both enrollment and latent search, is of secondary importance. The report will specify the hardware specifications used in the evaluation, and will also note that operational latent searching algorithms are likely to be implemented in more sophisticated hardware.

5 Latent Matching Software

5.1 Overview

Participants shall submit a set of SDKs (Software Development Kits) that provide the interfaces defined by the ELFT-EFS API specified below. The SDKs shall be provided as static or dynamic libraries to run on the NIST platform specified below. The ELFT-EFS API (Application Programmer Interface) is modeled after the API from ELFT Phase 2. The most notable differences from the ELFT Phase 2 API are that the exemplar and latent images and data provided to the SDK will be contained in ANSI/NIST files, and exemplar feature extraction will process a single exemplar per invocation (instead of the complete gallery). Also, the ELFT-EFS API specifies operational time limits on a per-processor core basis, rather than per-machine.

Each participant shall submit

- one SDK for exemplar feature extraction and exemplar enrollment
- one SDK for latent feature extraction
- one SDK for latent 1-to-N search

NIST recognizes the proprietary nature of the participant's software and will take all reasonable steps to protect this. The software submitted will be in an executable library format, and no algorithmic details need be supplied. NIST agrees not to use the Participant's software for purposes other than indicated above, without express permission by the Participant.

5.2 Test Platform

The NIST ELFT-EFS Evaluation test platform consists of an array of blade servers having a hardware configuration similar to:

Processor

- Dual 2.8 GHz/1MB Cache, Xeon (dual-core)
- 800 MHz Front Side Bus for PE 1855

Memory

- 16GB RAM (15GB available to applications)

Secondary storage

- 300GB 15K RPM Ultra SCSI Hard drives

The operating systems available (in order of preference) are:

- RedHat Enterprise Linux Server 5.1 (64-bit)
- Windows 2008 Server (64-bit)
- (Windows Server 32-bit may be available on request)

The available RAM for 64-bit SDKs will be no more than 15GB total. The available RAM for 32-bit SDKs will be no more than 3GB per process.

5.3 Execution protocol

Each SDK tested will be allocated multiple blades/cores from the array, along with a subset of the test data in order to maximize (time) efficiency through parallel operation.

Each SDK instance assigned to an individual blade or core will operate on a subset of the data, using individual data copies (as needed) from a local storage device.

For purposes of execution, there are two classes of SDKs, (1) sequential and (2) multithreaded. And each class the SDK may utilize either 32 or 64-bit execution mode. *Note that each SDK submitted (i.e. either of the two SDKs per participant) may be of a different class and execution mode. For example, the Exemplar feature extraction / enrollment SDK may be sequential 32-bit and the Latent feature extraction / search SDK may be multithreaded 64-bit.*

It is highly recommended that SDKs implement multithreading using 64-bit execution mode. However, if some participants are unable to submit multithreaded or 64-bit SDKs, we support other modes of operation as outlined below.

5.3.1 Sequential

An advantage of sequential (i.e. non-multithreaded) SDKs is the ability to "manually" parallelize SDK execution for a given test by executing multiple instances per blade server (e.g. one per core). A potential drawback is that individual 64-bit SDK instances have the potential to over-allocate available RAM, which may result in "swapping," decreasing overall execution speed. Another potential drawback is contention for resources given that each instance is executing independently (i.e. without coordinated resource usage). For this reason NIST does not recommend the submission of sequential SDKs.

As a simple example, the execution of a sequential SDK for a subtest requiring M latent searches against N exemplars (i.e. Gallery size N), may allocate M searches amongst K available cores such that each core is executing M/K searches total. The primary choice here is whether or not to allocate all cores available on a given blade server, or a subset thereof. How much memory is allocated by the SDK (limited by whether it is 32 or 64-bit mode) is a primary consideration.

Sequential SDKs which run in 32-bit execution mode shall have access to no more than 3GB per process. NIST will execute four (4) SDK instances (one instance per core) on each available blade server, in order to maximize processor and memory utilization.

Sequential SDKs which run in 64-bit execution mode shall have access of up to 15GB per process, and the participant should inform NIST at submission time as to the SDK's memory usage requirements. It is strongly recommended that the SDK perform most efficiently when executed as four (4) instances (one per core) on each blade server, where each instance allocates no more than a quarter of available RAM (i.e. 3.75GB), as opposed to when executed as a single (1) instance on each blade server which allocates all available RAM (i.e. 15GB). If more than 3.75GB is allocated per instance, the number of cores which can be utilized per blade server (without swapping) is essentially 15GB divided by the amount of RAM allocated per SDK instance (rounded to the nearest whole number).

5.3.2 Multithreaded

An advantage of multithreaded SDKs is the automatic utilization of available processor and memory resources through parallelization (without need for "manual" scheduling). Another advantage is coordinated access (of each thread) to resources such as disk I/O. For this reason NIST strongly recommends that submitted SDKs utilize multithreading aimed at maximizing usage of 4 cores and run in 64-bit mode in order to have access of up to 15GB of RAM.

As a simple example, the execution of a multithreaded SDK for a subtest requiring M latent searches of N exemplars (i.e. Gallery size N), will allocate M searches amongst K available blades such that each blade is executing M/K searches total.

Multithreaded SDKs which run in 64-bit mode have full access to all cores and memory (15GB) on each allocated blade. This approach clearly makes use of processing resources, and has the potential to mitigate contention issues through a coordinated use of parallelism.

Multithreaded SDKs which run in 32-bit mode will be limited to 3GB of RAM per process, which may limit their performance. Another option which exists here is for a multithreaded SDK to use no more than 2 threads, where each SDK instance uses the maximum 3GB of RAM. If informed, NIST could allocate two such SDKs per blade server in order to more fully utilize RAM.

5.4 API

5.4.1 Test Interface Description

Participants shall submit an SDK which provides the interfaces defined in section 5.4.4. Section 5.4.3 defines the interfaces to functions provided by NIST for use by the SDK. Sections 5.4.2 and 5.4.5 specify the declaration of constants, error codes, data-types and functions used by both.

The software undergoing testing will be hosted on NIST-supplied computers. The executable software under test will be built up from two sources: participant-supplied (SDKs) and NIST supplied (image extraction library and test driver).

5.4.2 Declarations

The following are declarations of data types and functions used in the Latent Fingerprint SDK testing interface:

```
////////////////////////////////////////////////////
// Declarations of constants                      //
////////////////////////////////////////////////////

// Impression type codes
#define IMPTYPE_LP      0     // Live-scan plain
#define IMPTYPE_LR      1     // Live-scan rolled
#define IMPTYPE_NP      2     // Nonlive-scan plain
#define IMPTYPE_NR      3     // Nonlive-scan rolled
```

```
// Finger position codes
#define FINGPOS_UK      0     // Unknown finger
#define FINGPOS_RT      1     // Right thumb
#define FINGPOS_RI      2     // Right index finger
#define FINGPOS_RM      3     // Right middle finger
#define FINGPOS_RR      4     // Right ring finger
#define FINGPOS_RL      5     // Right little finger
#define FINGPOS_LT      6     // Left thumb
#define FINGPOS_LI      7     // Left index finger
#define FINGPOS_LM      8     // Left middle finger
#define FINGPOS_LR      9     // Left ring finger
#define FINGPOS_LL      10    // Left little finger


////////////////////////////////////////////////////////////////
// Declarations for the NIST provided library functions       //
////////////////////////////////////////////////////////////////

// Structure to hold a single fingerprint record (image+metadata)
struct finger_record
{
      BYTE    impression_type;
      UINT16  resolution;     // Image resolution in pixels/cm
      BYTE    finger_position;
      UINT16  height;         // Image height in pixels
      UINT16  width;          // Image width in pixels
      BYTE    *image_data;    // 8-bit grayscale image data
};
typedef struct finger_record   FINGER_REC;

// Extracts 10 fingerprint records from a ten-print (AN2K) file
INT32 extract_image_data(     const    char    *tenprint_filename,
                              FINGER_REC **finger_recs);


// De-allocates the memory holding 10 fingerprint records
void free_image_data(FINGER_REC *finger_recs);


////////////////////////////////////////////////////////////////
// Declarations for the SDK provided library functions        //
////////////////////////////////////////////////////////////////


// Extracts features from exemplar
INT32 extract_exemplar( const char *exemplarFilename,
                        const char *outputDir);


// Creates a gallery from set of extracted exemplar features
INT32 create_gallery(   const INT32 numExemplars,
```

```
                        const char **exemplarFeatFilenames,
                        const char *galleryDir);

// Selects the current gallery for latent searching
INT32 set_gallery(      const char *galleryDir);

// Extracts features from latent file
INT32 extract_latent(   const char *latentFilename,
                        const char *outputDir);

// Searches for the latent in the gallery
INT32 latent_search(    const char *latentFeatFilename,
                        const char *outputDir);
```

5.4.3 NIST Provided Functions

5.4.3a Extract Image Data

INT32

```
extract_image_data(const char  *tenprint_filename,
                   FINGER_REC  **finger_recs);
```

Description

This function extracts ten fingerprint image records from a single (AN2K formatted) ten-print record file. The caller shall pass ***tenprint_filename*** as a pointer to the fully qualified pathname of an AN2K formatted ten-print record file, and ***finger_recs*** as the address of a pointer of type `FINGER_REC` (see 5.4.2 above).

Upon return ***finger_recs*** will contain a pointer to an array of ten `FINGER_REC` structures ordered by finger position from 1 (right thumb) to 10 (left little finger). For any fingers that are missing from the original ten-print record file, the *image_data* field in the respective `FINGER_REC` will be a NULL pointer.

Example

```
// Example of processing a ten-print record
FINGER_REC *finger_recs;
INT32 status=extract_image_data("E000123.an2", &finger_recs);
if (status == 0) {
        for (i=0;i<10;i++) {
                if (finger_recs[i].image_data != NULL)
                        process_valid_finger(finger_recs[i]);
                else
                        process_missing_finger(finger_recs[i]);
        }
        free_image_data(finger_recs); // see 5.4.3b below
}
```

Parameters

tenprint_filename (input): A pointer to a ten-print record filename.

finger_recs (output): The address of a FINGER_REC pointer.

Return Value

This function returns *zero* on success or a documented *non-zero* error code otherwise.

5.4.3b Free Image Data

```
void
free_image_data(FINGER_REC *finger_recs);
```

Description

De-allocates all memory used by the array of `FINGER_REC` structures specified by ***finger_recs*** which was allocated during a call to `extract_image_data()`.

Parameters

finger_recs (input): A pointer to an array of FINGER_REC structures.

Return Value

None.

5.4.4 SDK Provided Functions

5.4.4a Exemplar Feature Extraction

```
INT32
extract_exemplar( const char  *exemplarFilename,
                  const char   *outputDir);
```

Description

This function produces a single proprietary formatted feature set file from a 10-print exemplar set. The output from multiple calls to this function (i.e. multiple proprietary feature set files) will be used to construct a gallery (see section 5.4.4b) that is searchable by `latent_search()`.

The 10-print exemplar set will be contained in an ANSI/NIST file with pathname specified by ***exemplarFilename*** (e.g. "/mnt1/input/E1/E199999_1.an2"), and that file will contain either 10 rolled or 10 segmented slap fingerprint images. The directory to which the proprietary feature set file shall be written is specified by the pathname pointed to by ***outputDir*** (e.g. "/mnt/output/feats/E1/").