



Tattoo Recognition Technology - Evaluation (Tatt-E)

A Public Evaluation of Tattoo Recognition Algorithms

Concept, Evaluation Plan, and API Version 0.1

Mei Ngan and Patrick Grother

Contact via tatt-e@nist.gov



NIST
**National Institute of
Standards and Technology**
U.S. Department of Commerce

September 27, 2016

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

19

Provisional Timeline of the Tatt-E Activity

API Development	September 26, 2016	Draft evaluation plan available for public comments
	November 30, 2016	Final evaluation plan published
Phase 1	December 1, 2016	Participation starts: Algorithms may be sent to NIST
	February 14, 2017	Last day for submission of algorithms to Phase 1
	March 28, 2017	Interim results released to Phase 1 participants
Phase 2	May 15, 2017	Last day for submission of algorithms to Phase 2
	June 26, 2017	Interim results released to Phase 2 participants
Phase 3	August 31, 2017	Last day for submission of algorithms to Phase 3
	Q4 2017	Release of final public report

20

21

22

23

Acknowledgements

The organizers would like to thank the sponsor of this activity, the Federal Bureau of Investigation (FBI) Biometric Center of Excellence (BCOE) for initiating and progressing this work.

26

27

Contact Information

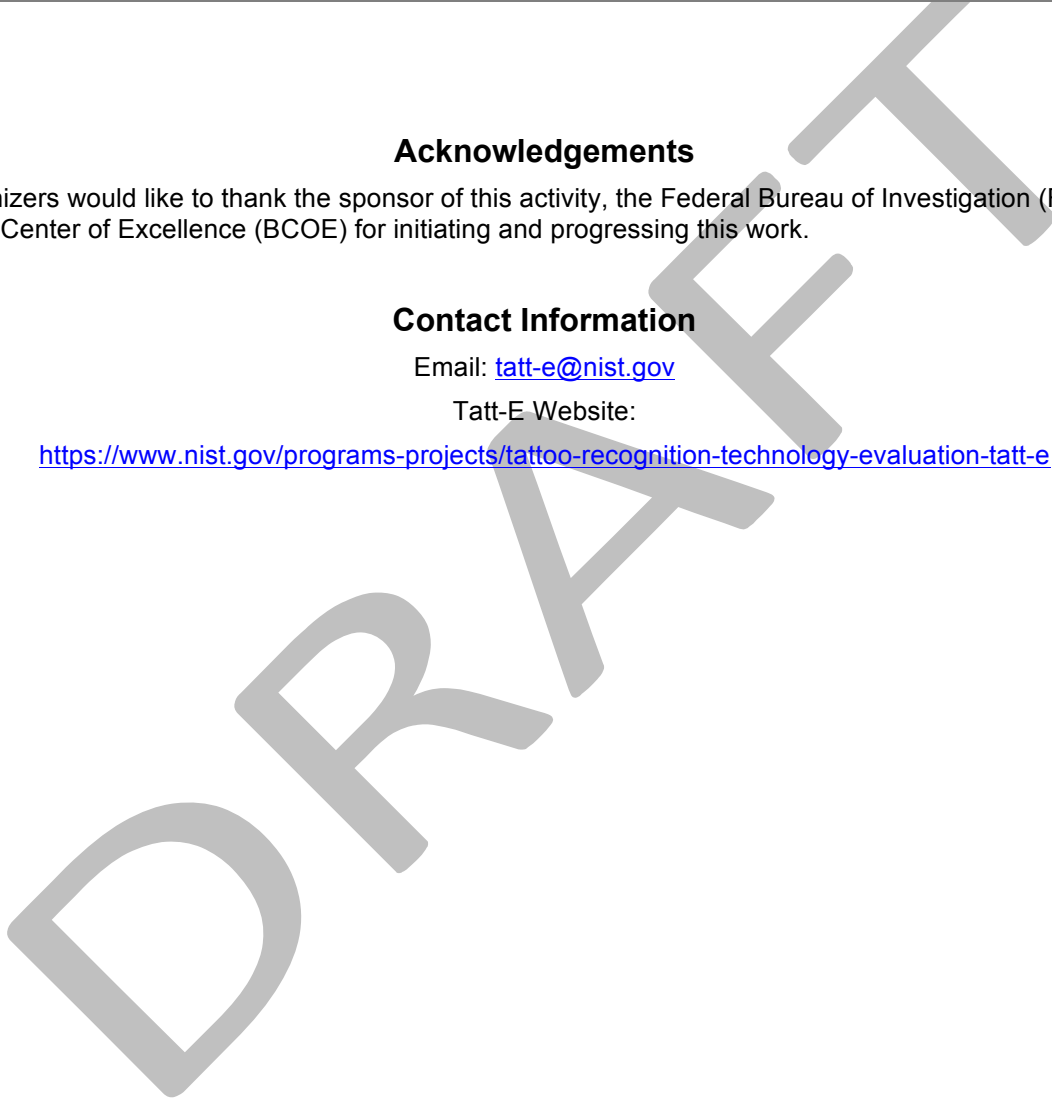
Email: tatt-e@nist.gov

Tatt-E Website:

<https://www.nist.gov/programs-projects/tattoo-recognition-technology-evaluation-tatt-e>

30

31



32 **Table of Contents**

33 1. Tatt-E..... 4

34 1.1 Background..... 4

35 1.2 The Tattoo Recognition Technology Program 4

36 1.3 Scope..... 5

37 1.4 Audience..... 5

38 1.5 Training Data 5

39 1.6 Offline Testing..... 5

40 1.7 Phased Testing 5

41 1.8 Interim reports..... 6

42 1.9 Final reports..... 6

43 1.10 Application scenarios..... 6

44 1.11 Rules for participation 7

45 1.12 Number and schedule of submissions..... 7

46 1.13 Core accuracy metrics 7

47 1.14 Reporting template size 7

48 1.15 Reporting computational efficiency..... 8

49 1.16 Exploring the accuracy-speed trade-space 8

50 1.17 Hardware specification 8

51 1.18 Operating system, compilation, and linking environment..... 8

52 1.19 Runtime behavior..... 10

53 1.20 Single-thread Requirement..... 10

54 1.21 Time limits..... 10

55 1.22 Ground truth integrity 11

56 2. Data structures supporting the API 12

57 2.1 Data structures 12

58 2.2 File structures for enrolled template collection 15

59 3. API Specification 15

60 3.1 Namespace..... 16

61 3.2 Overview..... 16

62 3.3 Detection and Localization (Class D)..... 16

63 3.4 Identification (Class I) 18

64 Annex A Submissions of Implementations to Tatt-E..... 23

65 A.1 Submission of implementations to NIST 23

66 A.2 How to participate..... 23

67 A.3 Implementation validation..... 24

68 Application and Agreement to Participate in the Tattoo Recognition Technology – Evaluation (Tatt-E)..... 25

70 **List of Tables**

71 Table 1 – Subtests supported under the Tatt-E activity..... 6

72 Table 2 – Tatt-E classes of participation..... 7

73 Table 3 – Cumulative total number of algorithms 7

74 Table 4 – Implementation library filename convention 9

75 Table 5 – Processing time limits in seconds, per 640 x 480 image 10

76 Table 6 – Enrollment dataset template manifest 15

77 Table 7 – Procedural overview of the detection and localization test 16

78 Table 8 – Procedural overview of the identification test 18

79

1. Tatt-E

1.1 Background

Tattoos have been used for many years to assist law enforcement in the identification of criminals and victims and for investigative research purposes. Historically, law enforcement agencies have followed the ANSI/NIST-ITL 1-2011¹ standard to collect and assign keyword labels to tattoos. This keyword labeling approach comes with drawbacks, which include the limited number of ANSI/NIST standard class labels to able describe the increasing variety of new tattoo designs, the need for multiple keywords to sufficiently describe some tattoos, and subjectivity in human annotation as the same tattoo can be labeled differently by examiners. As such, the shortcomings of keyword-based tattoo image retrieval have driven the need for automated image-based tattoo recognition capabilities.

1.2 The Tattoo Recognition Technology Program

The Tattoo Recognition Technology Program was initiated by NIST to support an operational need for image-based tattoo recognition to support law enforcement applications. The program provides quantitative support for tattoo recognition development and best practice guidelines. Program activities to date are summarized in Figure 1.

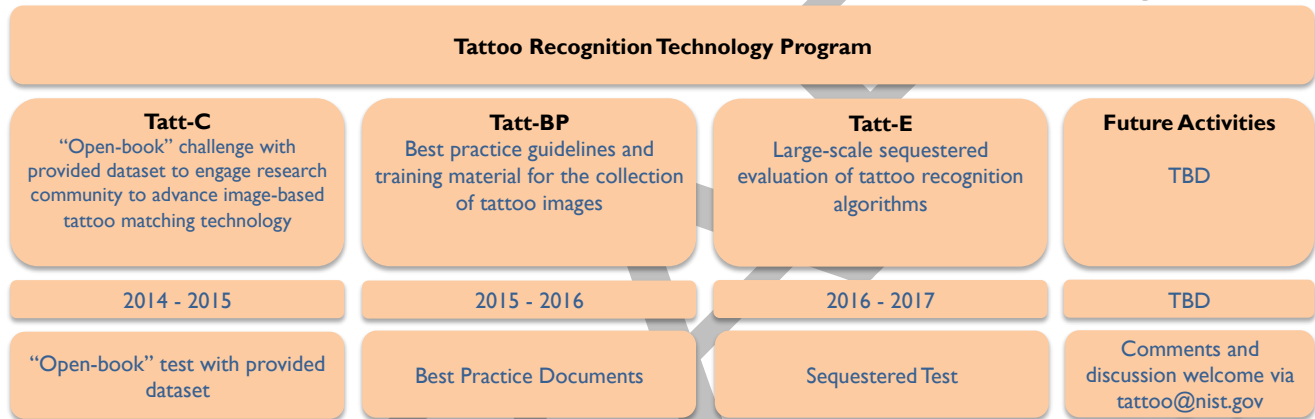


Figure 1 – Activities under the Tattoo Recognition Technology Program

- Tatt-C** was an initial research challenge that provided operational data and use cases to the research community to advance research and development into automated image-based tattoo technologies and to assess the state-of-the-art. NIST hosted a culminating industry workshop and published a public report on the outcomes and recommendations from the Tatt-C activity. Please visit <https://www.nist.gov/programs-projects/tattoo-recognition-technology-challenge-tatt-c> for more information.
- Tatt-BP** provides best practice guidance material for the proper collection of tattoo images to support image-based tattoo recognition. Recognition failure in Tatt-C was often related to the consistency and quality of image capture, and Tatt-BP aimed to provide guidelines on improving the quality of tattoo images collected operationally. Please visit <https://www.nist.gov/itl/iad/image-group/tattoo-recognition-technology-best-practices> for more information.
- Tatt-E** is a sequestered evaluation intended to assess tattoo recognition algorithm accuracy and run-time performance over a large-scale of operational data. The participation details of Tatt-E are established in this document, also available for download at <https://www.nist.gov/programs-projects/tattoo-recognition-technology-evaluation-tatt-e>.

¹ The latest version of the ANSI/NIST-ITL 1-2011 standard is available at <https://www.nist.gov/programs-projects/ansinist-itl-standard>.

115

116 1.3 Scope

117 The Tattoo Recognition Technology – Evaluation (Tatt-E) is being conducted to assess and measure the
118 capability of systems to perform automated image-based tattoo recognition. Both comparative and absolute
119 accuracy measures are of interest, given the goals to determine which algorithms are most effective and viable
120 for the following primary operational use-cases:

121

- 122 • Tattoo/Region of Interest Identification – matching different instances of the same tattoo image from the
123 same subject over time. This includes matching with entire and/or partial regions of a tattoo.
- 124 • Tattoo detection/localization – determining whether an image contains a tattoo and if so, segmentation
125 of the tattoo.
- 126 • Sketches – matching sketches to tattoo images.

127

128 **Out of scope:** Areas that are out of scope for this evaluation and will not be studied include: matching of tattoos
129 based on thematically similar content as the definition of “similarity” is ill-defined; tattoo recognition in video.

130

131 This document establishes a concept of operations and an application programming interface (API) for
132 evaluation of tattoo recognition implementations submitted to NIST’s Tattoo Recognition Technology –
133 Evaluation. See <https://www.nist.gov/programs-projects/tattoo-recognition-technology-evaluation-tatt-e> for all
134 Tatt-E documentation.

135 1.4 Audience

136 Any person or organizations with capabilities in any of the following areas are invited to participation in the Tatt-
137 E test.

138

- 138 • Tattoo matching implementations.
- 139 • Tattoo detection and localization algorithms.
- 140 • Algorithms with an ability to match sketches to tattoos.

141 Participants will need to implement the API defined in this document. Participation is open worldwide. There is
142 no charge for participation. NIST encourages submission of experimental prototypes as well as those that could
143 be readily made operational.

144 1.5 Training Data

145 None of the test data can be provided to participants. Instead prospective participants should leverage public
146 domain and proprietary datasets as available. The Tatt-C dataset, which is provided by the FBI, is a very
147 suitable tattoo corpus for development and training that has been made available to qualified developers -
148 please contact tatt-e@nist.gov for more details.

149 1.6 Offline Testing

150 While Tatt-E is intended as much as possible to mimic operational reality, this remains an offline test executed
151 on databases of images. The intent is to assess the core algorithmic capability of tattoo detection, localization,
152 and recognition algorithms. Offline testing is attractive because it allows uniform, fair, repeatable, and efficient
153 evaluation of the underlying technologies. Testing of implementations under a fixed API allows for a detailed set
154 of performance related parameters to be measured.

155 1.7 Phased Testing

156 To support development, Tatt-E will run in multiple phases. The final phase will result in the release of public
157 reports. Providers should not submit revised algorithms to NIST until NIST provides results for the prior phase.

158 For the schedule and number of algorithms of each class that may be submitted for each class, see section
159 1.12.

1.8 Interim reports

The performance of each implementation in phase 1 and 2 will be reported in a "report card". This will be provided to the participant. It is intended to facilitate research and development, not for marketing. Report cards will: be machine generated (i.e. scripted); be provided to participants with coded identification of their implementation; include timing, accuracy, and other performance results; include results from other implementations, but will not identify the other providers; be expanded and modified as revised implementations are tested and as analyses are implemented; be produced independently of the status of other providers' implementations; be regenerated on-the-fly, usually whenever any implementation completes testing, or when new analysis is added.

NIST does not intend to release these report cards publicly. NIST may release such information to the U.S. Government test sponsors; NIST will request that agencies not release this content.

1.9 Final reports

NIST will publish one or more final public reports. NIST may also publish: additional supplementary reports (typically as numbered NIST Interagency Reports); in academic journal articles; in conferences and workshops (typically PowerPoint).

Our intention is that the final test reports will publish results for the best-performing implementation from each participant. Because "best" is underdefined (accuracy vs. time, for example), the published reports may include results for other implementations. The intention is to report results for the most capable implementations (see section 1.13 on metrics). Other results may be included (e.g. in appendices) to show, for example, illustration of progress or tradeoffs.

IMPORTANT: All Phase 3 results will be attributed to the providers, publicly associating performance with organization name.

1.10 Application scenarios

As described in Table 1, the test is intended to represent:

- Use of tattoo recognition technologies in search applications in which the enrolled dataset could contain images in the hundreds of thousands.
- Tattoo detection and localization with zero or more tattoos in the sample.

Table 1 – Subtests supported under the Tatt-E activity

#	Class label	D	I
1.	Aspect	Detection and Localization	1:N Search
2.	Enrollment dataset	None, application to single images	N enrolled subjects
3.	Prior NIST test references	For detection task, see Detection in Tatt-C 2015 ²	See Tattoo Identification, Region of Interest, and Mixed Media matching from Tatt-C 2015 ²
4.	Example application	Database construction and maintenance of large amounts of unlabeled, comingled data, e.g. given a pile of seized media, 1. Detect whether/which images contain tattoos and 2. Segment tattoos as pre-processing step for search against a database.	Open-set search of a tattoo/sketch image against a central tattoo database, e.g. a search of a tattoo, parts of a tattoo, or a sketch of a tattoo against a tattoo database of known criminals.
5.	Number of images	Variable	Enrollment gallery: Up to O(10 ⁵)
6.	Number of images per individual	N/A	Variable: one or more still tattoo images
7.	Enrollment image types	Tattoo and non-tattoo images	Tattoos
8.	Probe image types	N/A	Tattoos and sketches

² See the Tatt-C test report: NIST Interagency Report 8078, linked from <https://www.nist.gov/programs-projects/tattoo-recognition-technology-challenge-tatt-c>

189

1.11 Rules for participation

There is no charge to participate in Tatt-E. A participant must properly follow, complete, and submit the Participation Agreement contained in this document. This must be done once, after December 1, 2016. It is not necessary to do this for each submitted software library.

- All participants shall submit at least one class D (detection and localization) algorithm.
- Class I (identification) algorithms may be submitted only if at least 1 class D algorithm is also submitted.
- All submissions shall implement exactly one of the functionalities defined in Table 2. A library shall not implement the API of more than one class (separate libraries shall be submitted to participate in separate participation classes).

199

Table 2 – Tatt-E classes of participation

Function	Detection and Localization	Identification
Class label	D	I
Co-requisite class	None	D
API requirements	3.3	3.4

200

1.12 Number and schedule of submissions

The test is conducted in three phases, as scheduled on page 2. The maximum total (i.e. cumulative) number of submissions is regulated in Table 3. Participation in Phase 1 is not required for algorithm submission in Phase 2.

205

Table 3 – Cumulative total number of algorithms

#	Phase 1	Total over Phases 1 + 2	Total over Phases 1 + 2 + 3
All classes of participation	2	4	6 if at least 1 was successfully executed by end of Phase 1 2 otherwise

1.13 Core accuracy metrics

For identification testing, the test will target open-universe applications such as searching tattoo databases of known criminals (where the subject may or may not exist in the gallery) and closed-set tasks where subject is known to be in the database, e.g. in prison or corrections environments. Both score-based and rank-based metrics will be considered. Rank-based metrics are appropriate for one-to-many applications that employ human examiners to adjudicate candidate lists. Score based metrics are appropriate for cases where transaction volumes are too high for human adjudication or when false alarm rates must otherwise be low. Metrics include, false positive and negative identification rate (FPIR and FNIR) and cumulative match characteristic that can depend on threshold and rank.

For detection and localization, assessments of overlap between detected and examiner-determined tattoo area will be considered along with score-based metrics including false positive and negative detection rate.

1.14 Reporting template size

Because template size is influential on storage requirements and computational efficiency, this API supports measurement of template size. NIST will report statistics on the actual sizes of templates produced by tattoo recognition implementations submitted to Tatt-E. NIST may also report statistics on runtime memory and other compute-performance characteristics.

221

222 1.15 Reporting computational efficiency

223 As with other tests, NIST will compute and report accuracy. In addition, NIST will also report timing statistics for
 224 all core functions of the submitted API implementations. This includes feature extraction and 1:N matching. For
 225 an example of how efficiency might be reported, see the final report of the FRVT 2013 test³.

226 1.16 Exploring the accuracy-speed trade-space

227 NIST will explore the accuracy vs. speed tradeoff for tattoo recognition algorithms running on a fixed platform.
 228 NIST will report both accuracy and speed of the implementations tested. While NIST cannot force submission
 229 of "fast vs. slow" variants, participants may choose to submit variants on some other axis (e.g. "experimental vs.
 230 mature") implementations.

231 1.17 Hardware specification

232 NIST intends to support highly optimized algorithms by specifying the runtime hardware. There are several
 233 types of computers that may be used in the testing. The following list gives some details about possible
 234 compute architectures:

- 235 • Dual Intel Xeon X5680 3.3 GHz CPUs (6 cores each)
- 236 • Dual Intel Xeon X7560 2.3 GHz CPUs (8 cores each)
- 237 • Dual Intel Xeon E5-2695 3.3 GHz CPUs (14 cores each; 56 logical CPUs total) with Dual NVIDIA Tesla
 238 K40 GPUs

239 Each CPU has 512K cache. The bus runs at 667 Mhz. The main memory is 192 GB Memory as 24 8GB
 240 modules. We anticipate that 16 processes can be run without time slicing, though NIST will handle all
 241 multiprocessing work *fork*. Participant-initiated multiprocessing is not permitted.

242 NIST is requiring use of 64-bit implementations throughout. This will support large memory allocation to support
 243 1:N identification tasks. Note that while the API allows read access of the disk during the 1:N search, the disk is
 244 relatively slow, and I/O will be included in your run time.

245 All GPU-enabled machines will be running CUDA version 7.5. cuDNN v5 for CUDA 7.5 will also be installed on
 246 these machines. Implementations that use GPUs will only be run on GPU-enabled machines.

247 1.18 Operating system, compilation, and linking environment

248 The operating system that the submitted implementations shall run on will be released as a downloadable file
 249 accessible from http://nigos.nist.gov:8080/evaluations/CentOS-7-x86_64-Everything-1511.iso, which is the 64-
 250 bit version of CentOS 7.2 running Linux kernel 3.10.0.

251 For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software
 252 must run under CentOS 7.2.

253 NIST will link the provided library file(s) to our C++ language test drivers. Participants are required to provide
 254 their library in a format that is dynamically-linkable using the C++11 compiler, g++ version 4.8.5.

255 A typical link line might be

256 `g++ -std=c++11 -l. -Wall -m64 -o tatte tatte.cpp -L. -ltatte_Company_D_07`

257 The Standard C++ library should be used for development. The prototypes from this document will be written to
 258 a file "tatte.h" which will be included via

```
#include <tatte.h>
```

259 The header files will be made available to implementers via <https://github.com/usnistgov/tattoo>.

260 All compilation and testing will be performed on x86_64 platforms. Thus, participants are strongly advised to
 261 verify library-level compatibility with g++ (on an equivalent platform) prior to submitting their software to NIST to
 262 avoid linkage problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file
 263 formats, etc.).

³ See the FRVT 2013 test report: NIST Interagency Report 8009, linked from <http://face.nist.gov/frvt>

264 Any and all dependencies on external dynamic/shared libraries not provided by CentOS 7.2 as part of the built-
 265 in “development” package must be provided as a part of the submission to NIST.

266 **1.18.1 Library and Platform Requirements**

267 Participants shall provide NIST with binary code only (i.e. no source code). The implementation should be
 268 submitted in the form of a dynamically-linked library file.

269 The core library shall be named according to Table 4. Additional dynamic libraries may be submitted that
 270 support this “core” library file (i.e. the “core” library file may have dependencies implemented in these other
 271 libraries).

272 Intel Integrated Performance Primitives (IPP) ® libraries are permitted if they are delivered as a part of the
 273 developer-supplied library package. It is the provider’s responsibility to establish proper licensing of all libraries.
 274 The use of IPP libraries shall not prevent run on CPUs that do not support IPP. Please take note that some IPP
 275 functions are multithreaded and threaded implementations are prohibited.

276 NIST will report the size of the supplied libraries.

277

278 **Table 4 – Implementation library filename convention**

Form	libTattE_provider_class_sequence.ending				
Underscore delimited parts of the filename	libTattE	provider	class	sequence	ending
Description	First part of the name, required to be this.	Single word name of the main provider EXAMPLE: Choice	Function classes supported in Table 2. EXAMPLE: D	A two digit decimal identifier to start at 00 and increment by 1 every time a library is sent to NIST. EXAMPLE: 07	.so
Example	libTattE_Choice_D_07.so				

279

280 **1.18.2 Configuration and developer-defined data**

281 The implementation under test may be supplied with configuration files and supporting data files. NIST will
 282 report the size of the supplied configuration files.

283 **1.18.3 Submission folder hierarchy**

284 Participant submissions should contain the following folders at the top level

- 285 • lib/ - contains all participant-supplied software libraries
- 286 • config/ - contains all configuration and developer-defined data
- 287 • doc/ - contains any participant-provided documentation regarding the submission
- 288 • validation/ - contains validation output

289 **1.18.4 Installation and Usage**

290 The implementation shall be installable using simple file copy methods. It shall not require the use of a separate
 291 installation program and shall be executable on any number of machines without requiring additional machine-
 292 specific license control procedures or activation. The implementation shall not use nor enforce any usage
 293 controls or limits based on licenses, number of executions, presence of temporary files, etc. It shall remain
 294 operable with no expiration date.

295 Hardware (e.g. USB) activation dongles are not acceptable.

296 **1.18.5 Modes of operation**

297 Implementations shall not require NIST to switch “modes” of operation or algorithm parameters. For example,
 298 the use of two different feature extractors must either operate automatically or be split across two separate
 299 library submissions.

300 1.19 Runtime behavior

301 1.19.1 Interactive behavior, stdout, logging

302 The implementation will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the
303 submitted library shall:

- 304 – Not use any interactive functions such as graphical user interface (GUI) calls, or any other calls, which
305 require terminal interaction e.g. reads from “standard input”.
- 306 – Run quietly, i.e. it should not write messages to “standard error” and shall not write to “standard output”.
- 307 – Only if requested by NIST for debugging, include a logging facility in which debugging messages are
308 written to a log file whose name includes the provider and library identifiers and the process PID.
309 Please do not enable this by default.

310 1.19.2 Exception Handling

311 The application should include error/exception handling so that in the case of a fatal error, the return code is still
312 provided to the calling application.

313 1.19.3 External communication

314 Processes running on NIST hosts shall not affect the runtime environment in any manner, except for memory
315 allocation and release. Implementations shall not write any data to external resource (e.g. server, file,
316 connection, or other process), nor read from such. If detected, NIST will take appropriate steps, including but not
317 limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and
318 documentation of the activity in published reports.

319 1.19.4 Stateless behavior

320 All components in this test shall be stateless. Thus, all functions should give identical output, for a given input,
321 independent of the runtime history. NIST will institute appropriate tests to detect stateful behavior. If detected,
322 NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from
323 the supplier, notification to the provider, and documentation of the activity in published reports.

324 1.20 Single-thread Requirement

325 Implementations must run in single-threaded mode, because NIST will parallelize the test by dividing the
326 workload across many cores and many machines simultaneously.

327 1.21 Time limits

328 The elemental functions of the implementations shall execute under the time constraints of Table 5. These time
329 limits apply to the function call invocations defined in Table 5. Assuming the times are random variables, NIST
330 cannot regulate the maximum value, so the time limits are 90-th percentiles. This means that 90% of all
331 operations should take less than the identified duration.

332 The time limits apply per image. When K tattoos images of a subject are present, the time limits shall be
333 increased by a factor K.

334 **Table 5 – Processing time limits (1 core) in seconds, per 640 x 480 image**

	D	I
Function	Detection and Localization	1:N identification
Feature extraction for enrollment and identification	5	5
Identification of one search template against 100,000 single-image tattoo records.		16
Enrollment finalization of 100,000 single-image tattoo records		720

(including disk IO time)		
--------------------------	--	--

335 **1.22 Ground truth integrity**

336 Some of the test data is derived from operational systems and may contain ground truth errors in which

- 337 – a single tattoo is present under two different identifiers, or
- 338 – two different tattoos are present under one identifier, or
- 339 – in which a tattoo is not present in the image.

340 If these errors are detected, they will be removed. NIST will use aberrant scores (high impostor scores, low
341 genuine scores) to detect such errors. This process will be imperfect, and residual errors are likely. For
342 comparative testing, identical datasets will be used and the presence of errors should give an additive increment
343 to all error rates. For very accurate implementations this will dominate the error rate. NIST intends to attach
344 appropriate caveats to the accuracy results. For prediction of operational performance, the presence of errors
345 gives incorrect estimates of performance.

346

DRAFT

347 2. Data structures supporting the API

348 2.1 Data structures

349 2.1.1 Overview

350 In this test, a tattoo is represented by $K \geq 1$ two-dimensional tattoo images.

351 2.1.2 Data structures for encapsulating multiple images

352 Some of the proposed datasets includes $K > 2$ same tattoo images per person for some persons. This affords
353 the possibility to model a recognition scenario in which a new image of a tattoo is compared against all prior
354 images. Use of multiple images per person has been shown to elevate accuracy over a single image for other
355 biometric modalities.

356 For tattoo recognition in this test, NIST will enroll $K \geq 1$ images for each unique tattoo. Both enrolled gallery and
357 probe samples may consist of multiple images such that a template is the result of applying feature extraction to
358 a set of $K \geq 1$ images and then integrating information from them. An algorithm might fuse K feature sets into a
359 single model or might simply maintain them separately. In any case the resulting proprietary template is
360 contained in a contiguous block of data. All identification functions operate on such multi-image templates.

361 The number of images per unique tattoo will vary, and images may not be acquired uniformly over time. NIST
362 currently estimates that the number of images K will never exceed 100. For the Tatt-E API, K of the same tattoo
363 images of an individual are contained in data structure of Section 2.1.2.2.

364 2.1.2.1 TattE::Image Struct Reference

365 Struct representing a single image.

366 Public Member Functions

- 367 • **Image** ()
- 368 • **Image** (uint16_t widthin, uint16_t heightin, uint8_t depthin, ImageType typein, std::shared_ptr<uint8_t>
369 datain)

370 Public Attributes

- 371 • **uint16_t width**
372 *Number of pixels horizontally.*
- 373 • **uint16_t height**
374 *Number of pixels vertically.*
- 375 • **uint16_t depth**
376 *Number of bits per pixel. Legal values are 8 and 24.*
- 377 • **ImageType imageType**
378 *Label describing the type of image.*
- 379 • **std::shared_ptr<uint8_t> data**
380 *Managed pointer to raster scanned data. Either RGB color or intensity. If image_depth == 24*
381 *this points to 3WH bytes RGBRGBRGB... If image_depth == 8 this points to WH bytes IIIIII.*

382 2.1.2.2 TattE::MultiTattoo Typedef Reference

383 typedef std::vector< Image > MultiTattoo

384 *Data structure representing a set of the same tattoo images from a single person.*

385 2.1.3 Data Structure for detected tattoo

386 Implementations shall return bounding box coordinates of each detected tattoo in an image.

387 2.1.3.1 TattE::BoundingBox Struct Reference

388 Structure for bounding box around a detected tattoo.

389 **Public Member Functions**

- 390 • **BoundingBox** ()
- 391 • **BoundingBox** (uint16_t xin, uint16_t yin, uint16_t widthin, uint16_t heightin, double confin)

392 **Public Attributes**

- 393 • uint16_t **x**
394 *X-coordinate of top-left corner of bounding box around tattoo.*
- 395 • uint16_t **y**
396 *Y-coordinate of top-left corner of bounding box around tattoo.*
- 397 • uint16_t **width**
398 *Width, in pixels, of bounding box around tattoo.*
- 399 • uint16_t **height**
400 *Height, in pixels, of bounding box around tattoo.*
- 401 • double **confidence**
402 *Certainty that this region contains a tattoo. This value shall be on [0, 1]. The higher the value,*
403 *the more certain.*

404 **2.1.4 Class for representing a tattoo in a MultiTattoo**405 **2.1.4.1 TattE::TattooRep Class Reference**

406 Class representing a tattoo or sketch template from image(s)

407 **Public Member Functions**

- 408 • **TattooRep** ()
409 *Default Constructor.*
- 410 • void **addBoundingBox** (const **BoundingBox** &bb)
411 *This function should be used to add bounding box entries for each input image provided to the*
412 *implementation for template generation. If there are 4 images in the MultiTattoo vector, then the*
413 *size of boundingBoxes shall be 4. boundingBoxes[i] is associated with MultiTattoo[i].*
- 414 • std::shared_ptr< uint8_t > **resizeTemplate** (uint64_t size)
415 *This function takes a size parameter and allocates memory of size and returns a managed*
416 *pointer to the newly allocated memory for implementation manipulation. This class will take care*
417 *of all memory allocation and de-allocation of its own memory. The implementation shall not de-*
418 *allocate memory created by this class.*
- 419 • const std::shared_ptr< uint8_t > **getTattooTemplatePtr** () const
- 420 • uint64_t **getTemplateSize** () const
421 *This function returns the size of the template data.*
- 422 • std::vector< **BoundingBox** > **getBoundingBoxes** () const
423 *This function returns the bounding boxes for detected tattoos associated with the input images.*

424 **Private Attributes**

- 425 • std::shared_ptr< uint8_t > **tattooTemplate**
426 *Proprietary template data representing a tattoo in images(s)*
- 427 • uint64_t **templateSize**
428 *Size of template.*
- 429 • std::vector< **BoundingBox** > **boundingBoxes**
430 *Data structure for capturing bounding boxes around the detected tattoo(s)*

431 **2.1.5 Data structure for result of an identification search**

432 All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted with
433 the most similar matching entries listed first with lowest rank.

434 **2.1.5.1 TattE::Candidate Struct Reference**

435 Data structure for result of an identification search.

436 **Public Member Functions**

- 437 • **Candidate** ()
- 438 • **Candidate** (bool assignedin, std::string idin, double scorein)

439 **Public Attributes**

- 440 • bool **isAssigned**
441 *If the candidate is valid, this should be set to true. If the candidate computation failed, this*
442 *should be set to false.*
- 443 • std::string **templateId**
444 *The template ID from the enrollment database manifest.*
- 445 • double **similarityScore**
446 *Measure of similarity between the identification template and the enrolled candidate. Higher*
447 *scores mean more likelihood that the samples are of the same person. An algorithm is free to*
448 *assign any value to a candidate. The distribution of values will have an impact on the*
449 *appearance of a plot of false-negative and false-positive identification rates.*

450 **2.1.6 Data Structure for return value of API function calls**451 **2.1.6.1 TattE::ReturnStatus Struct Reference**

452 A structure to contain information about the success/failure by the software under test. An object of this class
453 allows the software to return some information from a function call. The string within this object can be optionally
454 set to provide more information for debugging etc. The status code will be set by the function to Success on
455 success, or one of the other codes on failure.

456 **Public Member Functions**

- 457 • **ReturnStatus** ()
- 458 • **ReturnStatus** (const TattE::ReturnCode code, const std::string info="")
459 *Create a **ReturnStatus** object.*

460 **Public Attributes**

- 461 • TattE::ReturnCode **code**
462 *Return status code.*
- 463 • std::string **info**
464 *Optional information string.*

465 **2.1.7 Enumeration Type Documentation**466 **2.1.7.1 enum TattE::ReturnCode[strong]**

467 Return codes for the functions specified by this API.

468 **Enumerator**

- 469 **Success** Success
- 470 **ConfigError** Error reading configuration files
- 471 **ImageTypeNotSupported** Image type, e.g., sketches, is not supported by the implementation
- 472 **RefuseInput** Elective refusal to process the input
- 473 **ExtractError** Involuntary failure to process the image
- 474 **ParseError** Cannot parse the input data
- 475 **TemplateCreationError** Elective refusal to produce a template
- 476 **EnrollDirError** An operation on the enrollment directory failed (e.g. permission, space)

- 477 **NumDataError** The implementation cannot support the number of input images
- 478 **TemplateFormatError** One or more template files are in an incorrect format or defective
- 479 **InputLocationError** Cannot locate the input data - the input files or names seem incorrect
- 480 **VendorError** Vendor-defined failure

481 **2.1.7.2 enum TattE::TemplateRole[strong]**

482 Labels describing the type/role of the template to be generated (provided as input to template generation)

483 **Enumerator**

- 484 **Enrollment** Enrollment template used to enroll into gallery
- 485 **Identification** Identification template used for search

486 **2.1.7.3 enum TattE::ImageType[strong]**

487 Labels describing the image type.

488 **Enumerator**

- 489 **Tattoo** Tattoo image
- 490 **Sketch** Sketch of tattoo

492 **2.2 File structures for enrolled template collection**

493 An implementation converts a **MultiTattoo** into a template, using, for example the **createTemplate()** function of
 494 section 3.4.1.5.2. To support the Class I identification functions of Table 2, NIST will concatenate enrollment
 495 templates into a single large file, the EDB (for enrollment database). The EDB is a simple binary concatenation
 496 of proprietary templates. There is no header. There are no delimiters. The EDB may be hundreds of gigabytes
 497 in length.

498 This file will be accompanied by a manifest; this is an ASCII text file documenting the contents of the EDB. The
 499 manifest has the format shown as an example in **Error! Reference source not found..** If the EDB contains N
 500 templates, the manifest will contain N lines. The fields are space (ASCII decimal 32) delimited. There are three
 501 fields. Strictly speaking, the third column is redundant.

502 Important: If a call to the template generation function fails, or does not return a template, NIST will include the
 503 Template ID in the manifest with size 0. Implementations must handle this appropriately.

504 **Table 6 – Enrollment dataset template manifest**

Field name	Template ID	Template Length	Position of first byte in EDB
Datatype required	std::string	Unsigned decimal integer	Unsigned decimal integer
Example lines of a manifest file appear to the right. Lines 1, 2, 3 and N appear.	90201744	1024	0
	Tattoo01	1536	1024
	7456433	512	2560
	...		
	Tattoo12	1024	307200000

505 The EDB scheme avoids the file system overhead associated with storing millions of individual files.
 506

508 **3. API Specification**

509 The function prototypes from this document and any other supporting code will be provided in a "tatte.h" file
 510 made available to implementers via <https://github.com/usnistgov/tattoo>.

511 3.1 Namespace

512 All data structures and API interfaces/function calls will be declared in the `TattE` namespace.

513 3.2 Overview

514 This section describes separate APIs for the core tattoo applications described in section 1.10. All submissions
515 to Tatt-E shall implement the functions required by the rules for participation listed before Table 2. Tatt-E
516 participants shall implement the relevant C++ prototyped interfaces in this section. C++ was chosen in order to
517 make use of some object-oriented features.

518 3.3 Detection and Localization (Class D)

519 This section defines an API for algorithms that can solely perform tattoo detection and localization. The
520 detection task requires the implementation to detect whether an image contains a tattoo or not, and localization
521 requires identifying the location of the tattoo within the image. Given an image, an implementation should

- 522 • For detection, classify whether a tattoo was detected in the image or not and provide a real-valued
523 measure of detection confidence on $[0,1]$, with 1 indicating absolute certainty that the image contains a
524 tattoo and 0 indicating absolute certainty that the image does not contain a tattoo.
- 525 • For localization, report location(s) of one or more tattoos on different body locations in the form of a
526 bounding box.

527 **Table 7 – Procedural overview of the detection and localization test**

Phase	Name	Description	Performance Metrics to be reported by NIST
Detection and Localization	Initialization	initialize() Give the implementation the name of a directory where any provider-supplied configuration data will have been placed by NIST. This location will otherwise be empty. The implementation is permitted read-only access to the configuration directory.	
	Detection	detectTattoo() For each of N images, pass single images to the implementation for tattoo detection. The implementation will set a boolean indicating whether a tattoo was detected or not and a detection certainty confidence score. Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.	Statistics of detection times. Accuracy metrics. The incidence of where the implementation failed to perform detection (non-successful return code).
	Localization	localizeTattoos() For each of N tattoo images, pass single images to the implementation for tattoo localization. The implementation will populate a vector with bounding boxes corresponding to the tattoos detected from the input image. Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.	Statistics of the time needed for this operation. Accuracy metrics. The incidence of where the implementation failed to perform localization.

528

529 3.3.1 TattE::DetectAndLocalizeInterface Class Reference

530 The interface to Class D implementations.

531 3.3.1.1 Public Member Functions

- 532 • virtual `~DetectAndLocalizeInterface ()`
- 533 • virtual `ReturnStatus initialize (const std::string &configurationLocation)=0`

- 534 *This function initializes the implementation under test. It will be called by the NIST application*
 535 *before any call to the functions detectTattoo and localizeTattoos().*
- 536 • virtual **ReturnStatus detectTattoo** (const **Image** &inputImage, bool &tattooDetected, double
 537 &confidence)=0
 538 *This function takes an **Image** as input and indicates whether a tattoo was detected in the image*
 539 *or not.*
 - 540 • virtual **ReturnStatus localizeTattoos**(const **Image** &inputImage, std::vector< **BoundingBox** >
 541 &boundingBoxes)=0
 542 *This function takes an **Image** as input, and populates a vector of **BoundingBox** with the*
 543 *number of tattoos detected on different body locations from the input image.*

544 **3.3.1.2 Static Public Member Functions**

- 545 • static std::shared_ptr< **DetectAndLocalizeInterface** > **getImplementation** ()
 546 *Factory method to return a managed pointer to the **DetectAndLocalizeInterface***
 547 *object. This function is implemented by the submitted library and must return a managed*
 548 *pointer to the **DetectAndLocalizeInterface** object.*

549 **3.3.1.3 Detailed Description**

550 The interface to Class D implementations.
 551 The class D detection and localization software under test must implement the interface
 552 **DetectAndLocalizeInterface** by subclassing this class and implementing each method specified therein.

553 **3.3.1.4 Constructor & Destructor Documentation**

- 554 • virtual **TattE::DetectAndLocalizeInterface::DetectAndLocalizeInterface** ()[inline], [virtual]

555 **3.3.1.5 Member Function Documentation**

556 **3.3.1.5.1 virtual ReturnStatus TattE::DetectAndLocalizeInterface::initialize (const std::string &**
 557 **configurationLocation)[pure virtual]**

558 This function initializes the implementation under test. It will be called by the NIST application before any call to
 559 the functions detectTattoo and localizeTattoos.

560

561 **Parameters:**

in	<i>configurationLocation</i>	A read-only directory containing any developer-supplied configuration parameters or run-time data. The name of this directory is assigned by NIST, not hardwired by the provider. The names of the files in this directory are hardwired in the implementation and are unrestricted.
----	------------------------------	--

562 **3.3.1.5.2 virtual ReturnStatus TattE::DetectAndLocalizeInterface::detectTattoo (const Image &**
 563 **inputImage, bool & tattooDetected, double & confidence)[pure virtual]**

564 This function takes an **Image** as input and indicates whether a tattoo was detected in the image or not.

565

566 **Parameters:**

in	<i>inputImage</i>	An instance of an Image struct representing a single image
out	<i>tattooDetected</i>	true if a tattoo is detected in the image; false otherwise
out	<i>confidence</i>	A real-valued measure of tattoo detection confidence on [0,1]. A value of 1 indicates certainty that the image contains a tattoo, and a value of 0 indicates certainty that the image does not contain a tattoo.

567 **3.3.1.5.3 virtual ReturnStatus TattE::DetectAndLocalizeInterface::localizeTattoos(const Image &**
 568 **inputImage, std::vector< BoundingBox > & boundingBoxes, std::vector< BodyLocation > &**
 569 **bodyLocations)[pure virtual]**

570 This function takes an **Image** as input, and populates a vector of **BoundingBox** with the number of tattoos
 571 detected on different body locations from the input image.

572

573 **Parameters:**

in	<i>inputImage</i>	An instance of an Image struct representing a single image
out	<i>boundingBoxes</i>	For each tattoo detected in the image, the function shall create a BoundingBox , populate it with a confidence score, the x, y, width, height of the bounding box, and add it to the vector.

574 **3.3.1.6 static std::shared_ptr<DetectAndLocalizeInterface>**
 575 **TattE::DetectAndLocalizeInterface::getImplementation ()[static]**

576 Factory method to return a managed pointer to the **DetectAndLocalizeInterface** object.

577 This function is implemented by the submitted library and must return a managed pointer to the
 578 **DetectAndLocalizeInterface** object.

579 **Note:**

580 A possible implementation might be: `return (std::make_shared<ImplementationD> ());`

581 **3.4 Identification (Class I)**

582 The 1:N application proceeds in two phases, enrollment and identification. The identification phase includes
 583 separate pre-search feature extraction stage, and a search stage.

584 The design reflects the following *testing* objectives for 1:N implementations.

- support distributed enrollment on multiple machines, with multiple processes running in parallel
- allow recovery after a fatal exception, and measure the number of occurrences
- allow NIST to copy enrollment data onto many machines to support parallel testing
- respect the black-box nature of biometric templates
- extend complete freedom to the provider to use arbitrary algorithms
- support measurement of duration of core function calls
- support measurement of template size

585

Table 8 – Procedural overview of the identification test

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Enrollment	E1	Initialization	initializeEnrollmentSession() Give the implementation the name of a directory where any provider-supplied configuration data will have been placed by NIST. This location will otherwise be empty.	
	E2	Parallel Enrollment	createTemplate(TemplateRole=Enrollment) The input will be one or more of the same tattoo image. This function will pass the input to the implementation for conversion to a single template. The implementation will return a template to the calling application. NIST's calling application will be responsible for storing all templates as binary files. These will not be available to the implementation during this enrollment phase. Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers. The same tattoo will not be enrolled twice.	Statistics of the times needed to enroll a tattoo. Statistics of the sizes of created templates. The incidence of failed template creations.
	E3	Finalization	finalizeEnrollment() Permanently finalize the enrollment directory. This supports, for example, adaptation of the image-processing functions, adaptation of the representation, writing of a manifest, indexing, and	Size of the enrollment database as a function of population size N and the

			<p>computation of statistical information over the enrollment dataset.</p> <p>The implementation is permitted read-write-delete access to the enrollment directory during this phase.</p>	<p>number of images.</p> <p>Duration of this operation. The time needed to execute this function shall be reported with the preceding enrollment times.</p>
Pre-search	S1	Initialization	<p>initializeProbeTemplateSession()</p> <p>Tell the implementation the location of an enrollment directory. The implementation could look at the enrollment data. Implementation initialize in preparation for search template creation.</p> <p>The implementation is permitted read-only access to the enrollment directory during this phase.</p>	<p>Statistics of the time needed for this operation.</p> <p>Statistics of the time needed for this operation.</p>
	S2	Template preparation	<p>createTemplate(TemplateRole=Identification)</p> <p>For each probe, create a template from a set of input tattoo(s) or a sketch image. This operation will generally be conducted in a separate process invocation to step S3.</p> <p>The implementation is permitted no access to the enrollment directory during this phase.</p> <p>The result of this step is a search template.</p>	<p>Statistics of the time needed for this operation.</p> <p>Statistics of the size of the search template.</p>
Search	S3	Initialization	<p>initializeIdentificationSession()</p> <p>Tell the implementation the location of an enrollment directory. The implementation should read all or some of the enrolled data into main memory, so that searches can commence.</p> <p>The implementation is permitted read-only access to the enrollment directory during this phase.</p>	<p>Statistics of the time needed for this operation.</p>
	S4	Search	<p>identifyTemplate()</p> <p>A template is searched against the enrollment database.</p> <p>The implementation is permitted read-only access to the enrollment directory during this phase.</p>	<p>Statistics of the time needed for this operation.</p> <p>Accuracy metrics - Type I + II error rates.</p> <p>Failure rates.</p>

586

587 **3.4.1 TattE::IdentificationInterface Class Reference**

588 **3.4.1.1 Public Member Functions**

- 589 • virtual **~IdentificationInterface ()**
- 590 • virtual **ReturnStatus initializeEnrollmentSession** (const std::string &configurationLocation)=0
- 591 *This function initializes the implementation under test and sets all needed parameters.*
- 592 • virtual **ReturnStatus createTemplate** (const **MultiTattoo** &inputTattoos, const **TemplateRole**
- 593 **&templateType, TattooRep &tattooTemplate, double &quality)=0**
- 594 *This function takes a MultiTattoo and outputs a **TattooRep** object (essentially a template).*
- 595 • virtual **ReturnStatus finalizeEnrollment** (const std::string &enrollmentDirectory, const std::string
- 596 **&edbName, const std::string &edbManifestName)=0**
- 597 *This function will be called after all enrollment templates have been created and freezes the*
- 598 *enrollment data. After this call the enrollment dataset will be forever read-only.*
- 599 • virtual **ReturnStatus initializeProbeTemplateSession** (const std::string &configurationLocation, const
- 600 **std::string &enrollmentDirectory)=0**
- 601 *Before MultiTattoos are sent to the search template creation function, the test harness will call*
- 602 *this initialization function.*
- 603 • virtual **ReturnStatus initializeIdentificationSession** (const std::string &configurationLocation, const
- 604 **std::string &enrollmentDirectory)=0**
- 605 *This function will be called once prior to one or more calls to identifyTemplate. The function*
- 606 *might set static internal variables so that the enrollment database is available to the subsequent*
- 607 *identification searches.*

- 608 • virtual **ReturnStatus identifyTemplate** (const **TattooRep** &idTemplate, const uint32_t
609 candidateListLength, std::vector< **Candidate** > &candidateList)=0
610 *This function searches an identification template against the enrollment set, and outputs a*
611 *vector containing candidateListLength Candidates.*

612 3.4.1.2 Static Public Member Functions

- 613 • static std::shared_ptr< **IdentificationInterface** > **getImplementation** ()
614 *Factory method to return a managed pointer to the **IdentificationInterface** object.*
615

616 3.4.1.3 Detailed Description

617 The interface to Class I implementations.

618 The Class I submission software under test will implement this interface by subclassing this class and
619 implementing each method therein.

620 3.4.1.4 Constructor & Destructor Documentation

- 621 • virtual **TattE::IdentificationInterface::~IdentificationInterface** ()[inline], [virtual]
622

623 3.4.1.5 Member Function Documentation

624 3.4.1.5.1 virtual ReturnStatus TattE::IdentificationInterface::initializeEnrollmentSession (const 625 std::string & *configurationLocation*)[pure virtual]

626 This function initializes the implementation under test and sets all needed parameters.

627 This function will be called N=1 times by the NIST application, prior to parallelizing M >= 1 calls to
628 createTemplate() via fork().

629 Parameters:

in	<i>configurationLocation</i>	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
----	------------------------------	--

630 3.4.1.5.2 virtual ReturnStatus TattE::IdentificationInterface::createTemplate (const MultiTattoo & 631 *inputTattoos*, const TemplateRole & *templateType*, TattooRep & *tattooTemplate*, double & 632 *quality*)[pure virtual]

633 This function takes a MultiTattoo and outputs a **TattooRep** object (essentially a template).

634 For enrollment templates: If the function executes correctly (i.e. returns a successful exit status), the NIST
635 calling application will store the template. The NIST application will concatenate the templates and pass the
636 result to the enrollment finalization function. When the implementation fails to produce a template, it shall still
637 return a blank template (which can be zero bytes in length). The template will be included in the enrollment
638 database/manifest like all other enrollment templates, but is not expected to contain any feature information.

639 For identification templates: If the function returns a non-successful return status, the output template will be not
640 be used in subsequent search operations.

641 Parameters:

in	<i>inputTattoos</i>	An instance of a MultiTattoo structure. Implementations must alter their behavior according to the type and number of images/type of image contained in the structure. The input image type could be a tattoo or a sketch image. The MultiTattoo will always contain the same type of imagery, i.e., no mixing of tattoos and sketch images will occur. Note that implementation support for sketch images is OPTIONAL. Implementation shall return TattE::ImageType::ImageTypeNotSupported if they do not support sketch images. All algorithms must support tattoo images.
in	<i>templateType</i>	A value from the TemplateRole enumeration that indicates the intended usage of the template to be generated. In this case, either an enrollment template used for gallery enrollment or an identification template used for search.
out	<i>tattooTemplate</i>	Tattoo template object. For each tattoo detected in the MultiTattoo, the function shall

		provide the bounding box coordinates in each image. The bounding boxes shall be captured in the TattooRep.boundingBoxes variable, which is a vector of BoundingBox objects. If there are 4 images in the MultiTattoo vector, then the size of boundingBoxes shall be 4. boundingBoxes[i] is associated with MultiTattoo[i].
out	<i>quality</i>	A measure of tattoo quality on [0,1] indicative of expected utility to the matcher, or matchability. This value could measure tattoo distinctiveness/information richness, and would be an indicator of how well the tattoo would be expected to match. A value of 1 indicates high quality and that the tattoo would be expected to match well, and a value of 0 indicates low quality indicative that tattoo would not match well.

642 **3.4.1.5.3 virtual ReturnStatus TattE::IdentificationInterface::finalizeEnrollment (const std::string &**
 643 **enrollmentDirectory, const std::string & edbName, const std::string &**
 644 **edbManifestName)[pure virtual]**

645 This function will be called after all enrollment templates have been created and freezes the enrollment data.
 646 After this call the enrollment dataset will be forever read-only.

647 This function allows the implementation to conduct, for example, statistical processing of the feature data,
 648 indexing and data re-organization. The function may create its own data structure. It may increase or decrease
 649 the size of the stored data. No output is expected from this function, except a return code. The function will
 650 generally be called in a separate process after all the enrollment processes are complete. NOTE:
 651 Implementations shall not move the input data. Implementations shall not point to the input data.
 652 Implementations should not assume the input data would be readable after the call. Implementations must, **at a**
 653 **minimum, copy the input data** or otherwise extract what is needed for search.

654 **Parameters:**

in	<i>enrollmentDirectory</i>	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
in	<i>edbName</i>	The name of a single file containing concatenated templates, i.e. the EDB described in <i>Data Structures Supporting the API</i> . While the file will have read-write-delete permission, the implementation should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name. This is a NIST-provided input - implementers shall not internally hard-code or assume any values.
in	<i>edbManifestName</i>	The name of a single file containing the EDB manifest described in <i>Data Structures Supporting the API</i> . The file may be opened directly. It is not necessary to prepend a directory name. This is a NIST-provided input - implementers shall not internally hard-code or assume any values.

655 **3.4.1.5.4 virtual ReturnStatus TattE::IdentificationInterface::initializeProbeTemplateSession (const**
 656 **std::string & configurationLocation, const std::string & enrollmentDirectory)[pure virtual]**

657 Before MultiTattoos are sent to the search template creation function, the test harness will call this initialization
 658 function.

659 This function initializes the implementation under test and sets all needed parameters. This function will be
 660 called N=1 times by the NIST application, prior to parallelizing M >= 1 calls to createTemplate() via fork().
 661 Caution: The implementation should tolerate execution of P > 1 processes on the one or more machines each of
 662 which may be reading from this same enrollment directory in parallel. The implementation has read-only access
 663 to its prior enrollment data.

664 **Parameters:**

in	<i>configurationLocation</i>	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
in	<i>enrollmentDirectory</i>	The read-only top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.

665 **3.4.1.5.5 virtual ReturnStatus TattE::IdentificationInterface::initializeIdentificationSession (const**
 666 **std::string & configurationLocation, const std::string & enrollmentDirectory)[pure virtual]**

667 This function will be called once prior to one or more calls to identifyTemplate. The function might set static

668 internal variables so that the enrollment database is available to the subsequent identification searches.

669 **Parameters:**

in	<i>configurationLocation</i>	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
in	<i>enrollmentDirectory</i>	The read-only top-level directory in which enrollment data was placed.

670 3.4.1.5.6 **virtual ReturnStatus TattE::IdentificationInterface::identifyTemplate (const TattooRep &**
 671 ***idTemplate*, const uint32_t *candidateListLength*, std::vector< Candidate > &**
 672 ***candidateList*)[pure virtual]**

673 This function searches an identification template against the enrollment set, and outputs a vector containing
 674 candidateListLength Candidates.

675 Each candidate shall be populated by the implementation and added to candidateList. Note that candidateList
 676 will be an empty vector when passed into this function. The candidates shall appear in descending order of
 677 similarity score - i.e. most similar entries appear first.

678 **Parameters:**

in	<i>idTemplate</i>	A template from createTemplate() . If the value returned by that function was non-successful, the contents of idTemplate will not be used, and this function will not be called.
in	<i>candidateListLength</i>	The number of candidates the search should return.
out	<i>candidateList</i>	Each candidate shall be populated by the implementation. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.

679 3.4.1.5.7 **static std::shared_ptr<IdentificationInterface>**
 680 **TattE::IdentificationInterface::getImplementation ()[static]**

681
 682 Factory method to return a managed pointer to the **IdentificationInterface** object.

683 This function is implemented by the submitted library and must return a managed pointer to the
 684 **IdentificationInterface** object.

685 **Note:**

686 A possible implementation might be: `return (std::make_shared<ImplementationC>());`

687

688

689

690

Annex A

Submissions of Implementations to Tatt-E

A.1 Submission of implementations to NIST

NIST requires that all software, data and configuration files submitted by the participants be signed and encrypted. Signing is done with the participant's private key, and encryption is done with the NIST public key. The detailed commands for signing and encrypting are given here: <https://www.nist.gov/itl/iad/image-group/products-and-services/encrypting-softwaredata-transmission-nist>.

NIST will validate all submitted materials using the participant's public key, and the authenticity of that key will be verified using the key fingerprint. This fingerprint must be submitted to NIST by writing it on the signed participation agreement.

By encrypting the submissions, we ensure privacy; by signing the submission, we ensure authenticity (the software actually belongs to the submitter). NIST will reject any submission that is not signed and encrypted. NIST accepts no responsibility for anything that is transmitted to NIST that is not signed and encrypted with the NIST public key.

A.2 How to participate

Those wishing to participate in Tatt-E testing must do all of the following, on the schedule listed on Page 2.

- IMPORTANT: Follow the instructions for cryptographic protection of your software and data here - <https://www.nist.gov/itl/iad/image-group/products-and-services/encrypting-softwaredata-transmission-nist>.
- Send a signed and fully completed copy of the *Application to Participate in the Tattoo Recognition Technology - Evaluation (Tatt-E)* contained in this document. This must identify, and include signatures from, the Responsible Parties as defined in the application. The properly signed Tatt-E Application to Participate shall be sent to NIST as a PDF.
- Provide a software library that complies with the API (Application Programmer Interface) specified in this document.
 - Encrypted data and libraries below 20MB can be emailed to NIST at tatt-e@nist.gov.
 - Encrypted data and libraries above 20MB shall be
 - EITHER
 - Split into sections AFTER the encryption step. Use the unix "split" commands to make 9MB chunks, and then rename to include the filename extension need for passage through the NIST firewall.
 - `you% split -a 3 -d -b 9000000 libTattE_Choice_D_07.tgz.gpg`
 - `you% ls -l x??? | xargs -iQ mv Q libTattE_Choice_D_07_Q.tgz.gpg`
 - Email each part in a separate email. Upon receipt NIST will
 - `nist% cat tatte_choice_D07_*.tgz.gpg > libTattE_Choice_D_07.tgz.gpg`
 - OR
 - Made available as a file.zip.gpg or file.zip.asc download from a generic http webserver⁴,
 - OR
 - Mailed as a file.zip.gpg or file.zip.asc on CD / DVD to NIST at this address:

Tatt-E Test Liaison (A210) 100 Bureau Drive A210/Tech225/Stop 8940 NIST Gaithersburg, MD 20899-8940 USA	In cases where a courier needs a phone number, please use NIST shipping and handling on: 301 -- 975 -- 6296.
--	--

⁴ NIST will not register, or establish any kind of membership, on the provided website.

729 **A.3 Implementation validation**

730 Registered Participants will be provided with a small validation dataset and test program via
731 <https://github.com/usnistgov/tattoo> shortly after the final evaluation plan is released. An announcement will be
732 made on the Tatt-E website when the validation package is available.

733 The validation test programs shall be compiled by the provider. The output of these programs shall be
734 submitted to NIST.

735 Prior to submission of the software library and validation data, the Participant must verify that their software
736 executes on the validation images and produces correct scores and templates.

737 Software submitted shall implement the Tatt-E API Specification as detailed in the body of this document.

738 Upon receipt of the software library and validation output, NIST will attempt to reproduce the same output by
739 executing the software on the validation imagery, using a NIST computer. In the event of disagreement in the
740 output, or other difficulties, the Participant will be notified.

741

DRAFT

Application and Agreement to Participate in the Tattoo Recognition Technology – Evaluation (Tatt-E)

Last Updated: September 26, 2016

1. Who Should Participate

- 1.1. Tattoo recognition technology researchers and developers from industry, research institutions, and academia are eligible to participate in the Tattoo Recognition Technology - Evaluation (Tatt-E) – hereafter referred to as the “Tatt-E”.
- 1.2. Anonymous participation will not be permitted. This means that signatories to this document, Tattoo Recognition Technology – Evaluation – Application to Participate (“Agreement”), acknowledge that they understand that the results (see Section 6) of the test of the Submission will be published with attribution to their Organization.

2. How to Participate

- 2.1. In order to participate in Tatt-E, an Organization must provide the information requested in Section 8 of this Agreement identifying the Responsible Party and the Point of Contact. Organization must also print and sign this Agreement, attach business cards from each of the signing parties, and send it to the location designated in Section 8. Signatures of both the Responsible Party and the Point of Contact are required.
 - 2.1.1. The Responsible Party is an individual with the authority to commit the organization to the terms in this Agreement.
 - 2.1.2. The Point of Contact (POC) is an individual with detailed knowledge of the participating Submission.
 - 2.1.3. In some cases, the Responsible Party and the POC may be the same person.
- 2.2. Upon receipt of the signed application by the National Institute of Standards and Technology (NIST), the organization will be classified as a “Tentative Evaluation Participant.” NIST must receive this signed application with the algorithm prototypes. Algorithm prototypes shall be submitted as pre-compiled software libraries. They may be submitted during the submission period from **December 1, 2016 to August 31, 2017**. The application is required to be submitted with the **first** software library submission; subsequent submissions do not require additional applications.
- 2.3. It is the Government’s desire to select all Tentative Participants as Participants. However, if demand for participation exceeds the Government’s ability to properly evaluate the technology, the Government will select Participants on a first come - first served basis.
- 2.4. Participant shall provide a submission (“Submission”), as specified in the document *Tatt-E: Concept, Evaluation Plan, and API (“Test Plan”)* available at <https://www.nist.gov/itl/iad/image-group/programsprojects/tattoo/tattoo-recognition-technology-evaluation-tatt-e>. A Submission shall include all executable code, validation results, configuration files, documentation, and all other files required by NIST and the Participant to validate and execute the tests specified in the Test Plan.
- 2.5. The Submission need not be used in a production system or be commercially available. However, the Submission must, at a minimum, be a stable implementation capable of conforming to the Test Plan that NIST has published for Tatt-E.
- 2.6. The Submission must be encrypted before transmitting to NIST. Instructions for Submission can be found on the Tatt-E website. Generic encryption instructions can be found in the Image Group’s *Encrypting Software for Transmission to NIST* document available at <https://www.nist.gov/itl/iad/image-group/products-and-services/encrypting-softwaredata-transmission-nist>. A box for the Participant’s public key fingerprint is included on the Agreement. Submissions that are not signed with the public key fingerprint listed on the Agreement will not be accepted.
- 2.7. Submissions must be compliant with the Test Plan, NIST test hardware, and NIST test software.

788 Submissions must be delivered to NIST during the submission period given in paragraph 2.2 according
789 to the technical specifications given in the Test Plan.

790 3. Points of Contact

791 3.1. The Tatt-E Liaison is the U.S. Government point of contact for Tatt-E.

792 3.2. All questions should be directed to tatt-e@nist.gov, which will be received by the Tatt-E Liaison and
793 other Tatt-E personnel.

794 3.3. These questions and answers maybe provided as updates to the *Tatt-E: Concept, Evaluation Plan, and*
795 *API* at the discretion of the Tatt-E Liaison.

796 4. Release of Tatt-E 2017 Results

797 4.1. After the completion of Tatt-E testing, the U.S. Government will publish all results obtained, along with
798 the Organization's name in Final Report(s).

799 4.2. Participant will be notified of their results via the Responsible Party and the Point of Contact provided
800 on the Agreement.

801 4.3. After the release of Tatt-E results, Participant may use the results for their own purposes. Such results
802 shall be accompanied by the following phrase: "Results show from NIST do not constitute an
803 endorsement of any particular system, product, service, or company by the U.S. Government." Such
804 results shall also be accompanied by the Internet address (URL) of the Tatt-E website
805 ([https://www.nist.gov/itl/iad/image-group/programsprojects/tattoo/tattoo-recognition-technology-
806 evaluation-tatt-e](https://www.nist.gov/itl/iad/image-group/programsprojects/tattoo/tattoo-recognition-technology-evaluation-tatt-e)).

807 5. Additional Information

808 5.1. Any data obtained during Tatt-E, as well as any documentation required by the U.S. Government from
809 the Participant (except the Submission), becomes the property of the U.S. Government. Participant will
810 not acquire a proprietary interest in the data and/or submitted documentation. The data and
811 documentation will be treated as sensitive information and only be used for the purposes of the Tatt-E
812 test.

813 5.2. Participant agrees that they not file any Tatt-E-related claim against Tatt-E sponsors, supporters, staff,
814 contractors, or agency of the U.S. Government, or otherwise seek compensation for any equipment,
815 materials, supplies, information, travel, labor and/or other Participant-provided services.

816 5.3. The U.S. Government is not bound or obligated to follow any recommendations that may be submitted
817 by the Participant. The U.S. Government, or any individual agency, is not bound, nor is it obligated, in
818 any way to give any special consideration to Participant on future contracts.

819 5.4. NIST is conducting Tatt-E pursuant to 15 U.S.C. §272(b)(8), (c)(2), and (c)(14).

820 5.5. By signing this Agreement, Participant acknowledges that they understand any test details and/or
821 modifications that are provided on the Tatt-E website supersede the information in this Agreement.

822 5.6. Participant may withdraw from Tatt-E at any time before their Submission is received by NIST, without
823 their participation and withdrawal being documented in the Final Report(s).

824 5.7. NIST will use the Participant's Submission only for the agreed-upon Tatt-E test, and in the event errors
825 are subsequently found, to re-run prior tests and resolve those errors.

826 5.8. NIST agrees not to use the Participant's Submission for purposes other than indicated above, without

Tattoo Recognition Technology - Evaluation (Tatt-E)

827 express permission by the Participant.

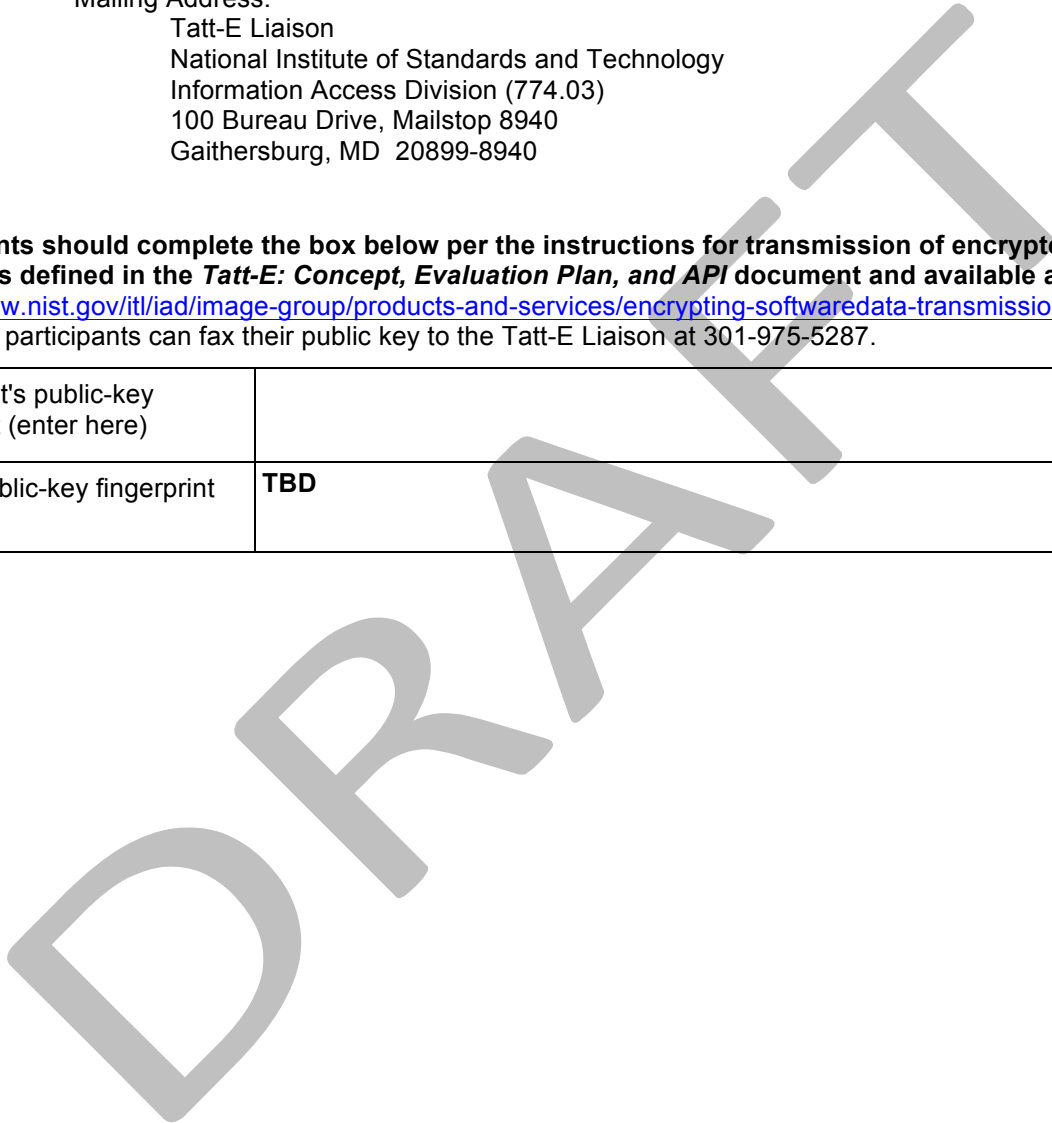
828 5.9. Original signed copies of the Tatt-E application are required. Original, signed copies of this application,
829 with business cards from both signing parties attached, must be mailed to the address below. These
830 must be signed paper hardcopies. Scanned documents submitted via email are not acceptable. Please
831 send an email message to tatt-e@nist.gov stating that you have sent your application. NIST will not
832 accept applications from generic email addresses (e.g. gmail.com, hotmail.com, etc.). Upon receipt of
833 your application, we will send you a confirmation email message.

834 Mailing Address:
835 Tatt-E Liaison
836 National Institute of Standards and Technology
837 Information Access Division (774.03)
838 100 Bureau Drive, Mailstop 8940
839 Gaithersburg, MD 20899-8940
840
841

842 **Participants should complete the box below per the instructions for transmission of encrypted content**
843 **to NIST as defined in the *Tatt-E: Concept, Evaluation Plan, and API* document and available at**
844 **<https://www.nist.gov/itl/iad/image-group/products-and-services/encrypting-softwaredata-transmission-nist>.** If
845 preferred, participants can fax their public key to the Tatt-E Liaison at 301-975-5287.

Participant's public-key fingerprint (enter here)	
NIST's public-key fingerprint	TBD

846



847 **Request to Participate**

848 **With my signature**, I hereby request consideration as a Participant in the Tattoo Recognition Technology -
849 Evaluation (Tatt-E), and I am authorizing my Organization to participate in Tatt-E according to the rules and
850 limitations listed in this Agreement.

851

852 **With my signature**, I also state that I have the authority to accept the terms stated in this Agreement.

853

854

855

856

857 _____
SIGNATURE, TITLE AND ORGANIZATION OF RESPONSIBLE PARTY

DATE

858

859

860

861 _____
PRINTED NAME AND EMAIL ADDRESS OF RESPONSIBLE PARTY

862

863

864

865

866

867 _____
SIGNATURE, TITLE AND ORGANIZATION OF POINT OF CONTACT

DATE

868

869

870

871 _____
PRINTED NAME AND EMAIL ADDRESS OF POINT OF CONTACT

872

873

874

875

876

877

878

879

880

881

882

883 _____
ATTACH BUSINESS CARDS HERE FOR ALL SIGNING PARTIES

884

