#### 2.1.5.1 TattE::Candidate Struct Reference

Data structure for result of an identification search.

**Public Member Functions**

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

**Public Attributes**

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

### 2.1.6 Data Structure for return value of API function calls

#### 2.1.6.1 TattE::ReturnStatus Struct Reference

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

**Public Member Functions**

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

**Public Attributes**

- TattE::ReturnCode **code**
  *Return status code.*
- std::string **info**
  *Optional information string.*

### 2.1.7 Enumeration Type Documentation

#### 2.1.7.1 enum TattE::ReturnCode[strong]

Return codes for the functions specified by this API.

**Enumerator**

***Success*** Success

***ConfigError*** Error reading configuration files

***ImageTypeNotSupported*** Image type, e.g., sketches, is not supported by the implementation

***RefuseInput*** Elective refusal to process the input

***ExtractError*** Involuntary failure to process the image

***ParseError*** Cannot parse the input data

***TemplateCreationError*** Elective refusal to produce a template

***EnrollDirError*** An operation on the enrollment directory failed (e.g. permission, space)

***NumDataError*** The implementation cannot support the number of input images

***TemplateFormatError*** One or more template files are in an incorrect format or defective

***InputLocationError*** Cannot locate the input data - the input files or names seem incorrect

***VendorError*** Vendor-defined failure

#### 2.1.7.2 enum TattE::TemplateRole[strong]

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

**Enumerator**

***Enrollment*** Enrollment template used to enroll into gallery

***Identification*** Identification template used for search

#### 2.1.7.3 enum TattE::ImageType[strong]

Labels describing the image type.

**Enumerator**

***Tattoo*** Tattoo image

***Sketch*** Sketch of tattoo

## 2.2 File structures for enrolled template collection

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

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

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

**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 |

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

# 3. API Specification

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

## 3.1 Namespace

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

## 3.2 Overview

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

## 3.3 Detection and Localization (Class D)

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

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

**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()**<br>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()**<br>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.<br>Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers. | Statistics of detection times.<br>Accuracy metrics.<br>The incidence of where the implementation failed to perform detection (non-successful return code). |
| | Localization | **localizeTattoos()**<br>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.<br>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.<br>Accuracy metrics.<br>The incidence of where the implementation failed to perform localization. |

### 3.3.1 TattE::DetectAndLocalizeInterface Class Reference

The interface to Class D implementations.

#### 3.3.1.1 Public Member Functions

- virtual **~DetectAndLocalizeInterface** ()
- virtual **ReturnStatus initialize** (const std::string &configurationLocation)=0

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

- virtual **ReturnStatus detectTattoo** (const **Image** &inputImage, bool &tattooDetected, double &confidence)=0

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

- virtual **ReturnStatus localizeTattoos**(const **Image** &inputImage, std::vector< **BoundingBox** > &boundingBoxes)=0

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

#### 3.3.1.2 Static Public Member Functions

- static std::shared_ptr< **DetectAndLocalizeInterface** > **getImplementation** ()

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

#### 3.3.1.3 Detailed Description

The interface to Class D implementations.

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

#### 3.3.1.4 Constructor & Destructor Documentation

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

#### 3.3.1.5 Member Function Documentation

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

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

**Parameters:**

| | | |
|---|---|---|
| in | *configurationLocation* | 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. |

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

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

**Parameters:**

| | | |
|---|---|---|
| in | *inputImage* | An instance of an **Image** struct representing a single image |
| out | *tattooDetected* | true if a tattoo is detected in the image; false otherwise |
| out | *confidence* | 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. |

##### 3.3.1.5.3 virtual ReturnStatus TattE::DetectAndLocalizeInterface::localizeTattoos(const Image & inputImage, std::vector< BoundingBox > & boundingBoxes, std::vector< BodyLocation > & 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 | *inputImage* | An instance of an **Image** struct representing a single image |
| out | *boundingBoxes* | 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()**<br>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)**<br>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.<br>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.<br>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.<br>Statistics of the sizes of created templates.<br>The incidence of failed template creations. |
| | E3 | Finalization | **finalizeEnrollment()**<br>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 |

| | | | | |
|---|---|---|---|---|
| | | | computation of statistical information over the enrollment dataset.<br>The implementation is permitted **read-write-delete access** to the enrollment directory during this phase. | number of images.<br>Duration of this operation. The time needed to execute this function shall be reported with the preceding enrollment times. |
| Pre-search | S1 | Initialization | **initializeProbeTemplateSession()**<br>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.<br>The implementation is permitted **read-only access** to the enrollment directory during this phase. | Statistics of the time needed for this operation.<br>Statistics of the time needed for this operation. |
| | S2 | Template preparation | **createTemplate(TemplateRole=Identification)**<br>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.<br>The implementation is **permitted no access** to the enrollment directory during this phase.<br>The result of this step is a search template. | Statistics of the time needed for this operation.<br>Statistics of the size of the search template. |
| Search | S3 | Initialization | **initializeIdentificationSession()**<br>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.<br>The implementation is permitted **read-only access** to the enrollment directory during this phase. | Statistics of the time needed for this operation. |
| | S4 | Search | **identifyTemplate()**<br>A template is searched against the enrollment database.<br>The implementation is permitted **read-only access** to the enrollment directory during this phase. | Statistics of the time needed for this operation.<br>Accuracy metrics - Type I + II error rates.<br>Failure rates. |

### 3.4.1 TattE::IdentificationInterface Class Reference

#### 3.4.1.1 Public Member Functions

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

- virtual **ReturnStatus identifyTemplate** (const **TattooRep** &idTemplate, const uint32_t candidateListLength, std::vector< **Candidate** > &candidateList)=0

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

#### 3.4.1.2 Static Public Member Functions

- static std::shared_ptr< **IdentificationInterface** > **getImplementation** ()

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

#### 3.4.1.3 Detailed Description

The interface to Class I implementations.

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

#### 3.4.1.4 Constructor & Destructor Documentation

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

#### 3.4.1.5 Member Function Documentation

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

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

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

**Parameters:**

| | | |
|---|---|---|
| in | *configurationLocation* | A read-only directory containing any developer-supplied configuration parameters or run-time data files. |

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

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

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

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

**Parameters:**

| | | |
|---|---|---|
| in | *inputTattoos* | 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 | *templateType* | 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 | *tattooTemplate* | 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 | *quality* | 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 would not match well. |

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

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

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

**Parameters:**

| | | |
|---|---|---|
| in | *enrollmentDirectory* | 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 | *edbName* | The name of a single file containing concatenated templates, i.e. the EDB described in *Data Structures Supporting the API*. 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 | *edbManifestName* | The name of a single file containing the EDB manifest described in *Data Structures Supporting the API* . 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. |

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

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

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

**Parameters:**

| | | |
|---|---|---|
| in | *configurationLocation* | A read-only directory containing any developer-supplied configuration parameters or run-time data files. |
| in | *enrollmentDirectory* | 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. |

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

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

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

**Parameters:**

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

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

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

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

**Parameters:**

| | | |
|---|---|---|
| in | *idTemplate* | 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 | *candidateListLength* | The number of candidates the search should return. |
| out | *candidateList* | 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. |

#### 3.4.1.5.7 static std::shared_ptr<IdentificationInterface> TattE::IdentificationInterface::getImplementation ()[static]

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

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

**Note:**

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

# 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 -1 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 webserver[4],

      OR
        - Mailed as a file.zip.gpg or file.zip.asc on CD / DVD to NIST at this address:

| | |
|---|---|
| Tatt-E Test Liaison (A210)<br>100 Bureau Drive<br>A210/Tech225/Stop 8940<br>NIST<br>Gaithersburg, MD 20899-8940<br>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.

## A.3 Implementation validation

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

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

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

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

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

# 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.

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

## 3. Points of Contact

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

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

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

## 4. Release of Tatt-E 2017 Results

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

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

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

## 5. Additional Information

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

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

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

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

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

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

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

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

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

## Request to Participate

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

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

_____ _____

SIGNATURE, TITLE AND ORGANIZATION OF RESPONSIBLE PARTY DATE

_____

PRINTED NAME AND EMAIL ADDRESS OF RESPONSIBLE PARTY

_____ _____

SIGNATURE, TITLE AND ORGANIZATION OF POINT OF CONTACT DATE

_____

PRINTED NAME AND EMAIL ADDRESS OF POINT OF CONTACT

_____

ATTACH BUSINESS CARDS HERE FOR ALL SIGNING PARTIES