Face Recognition Vendor Test
Ongoing

Performance of Automated
Presentation Attack Detection (PAD) Algorithms
Concept, Evaluation Plan, and API

VERSION 0.1

Mei Ngan
Patrick Grother
Kayee Hanaoka
Austin Hom
Joyce Yang
Information Access Division
Information Technology Laboratory

March 23, 2022

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

Revision History

Date	Version	Description
March 23, 2022	0.1	Draft document for public comment

Table of Contents

List of Tables

1. PAD

1.1. Scope

A presentation attack (PA), as defined by the ISO/IEC 30107[1] standard on biometric presentation attack detection, is "the presentation of an artefact or of human characteristics to a biometric capture subsystem in a fashion intended to interfere with system policy". A presentation attack is often launched with the intent of *impersonation* (the user is trying to authenticate as a target identity) or *evasion* (the user is trying to fool the biometric system into not recognizing their true identity). The goals of impersonation include trying to gain positive access privilege as someone else, for example, trying to unlock someone's cell phone or gain access to a facility. The goals of evasion are typically to conceal one's true identity to evade recognition from say a watchlist, or to create a separate enrollment under a different name. Biometric systems can potentially be attacked by an unknown number of presentation attack instruments, and the number or type of attack instruments in existence is not well-known. Some examples of known presentation attack instruments include artificial "gummy" fingers[2], "replay" attacks where the attacker is holding a photo or video of someone's face to the camera[3], and iris photo and contact lens attacks[4].

Presentation attack of face recognition systems (and the ability to detect it) is an area of high interest given the widespread deployment of face recognition systems, particularly in unmanned/unsupervised and remote enrollment and authentication scenarios. PAD capabilities generally fall into two categories – software-based and hardware-based PAD. Table 1 summarizes the relevance and applications of software vs. hardware-based PAD. The FRVT PAD test will provide ongoing independent testing of software-based facial PAD detection technologies. The evaluation is designed to assess software-based PA detection capability to inform developers and current and prospective end-users. Software-based PAD solutions operate only on the captured imagery. This document establishes an initial concept of operations and an application programming interface (API) for evaluation of algorithmic capability to detect facial presentation attack from still photographs and/or video frames.

Note: Hardware-based PAD solutions are currently out of scope in FRVT PAD. For developers interested in evaluation of hardware-based PAD capabilities, the DHS Science and Technology Directorate is planning a future technology demonstration to include testing of hardware-based PAD capabilities.

Table 1 – Software vs. hardware-based PAD

	Software-based PAD	Hardware-based PAD
Input	Image/video	Image/video + other non-standardized data or signals sensed by dedicated hardware
Mode of operation	Server-based PAD with non-face-aware capture device; offline PAD in existing/legacy systems	Client or edge-based PAD with dedicated face-aware capture device
Applications	Applications where capture processes and devices are not controlled or cannot be configured to perform PAD	Applications where hardware is controllable/configurable during the capture process to perform PAD

[1] ISO/IEC 30107-1:2016 Information technology — Biometric presentation attack detection — Part 1: Framework
[2] Tsutomu Matsumoto, Hiroyuki Matsumoto, Koji Yamada, and Satoshi Hoshino "Impact of artificial "gummy" fingers on fingerprint systems", Proc. SPIE 4677, Optical Security and Counterfeit Deterrence Techniques IV, (19 April 2002); https://doi.org/10.1117/12.462719
[3] https://mobidev.biz/blog/face-anti-spoofing-prevent-fake-biometric-detection
[4] Chaos Computer Club Berlin: Chaos Computer Clubs breaks iris recognition system of the Samsung Galaxy S8 (2017). https://www.ccc.de/en/updates/2017/iriden

1.2. General FRVT Evaluation Specifications

General and common information shared between all Ongoing FRVT tracks are documented in the FRVT General Evaluation Specifications document – https://pages.nist.gov/frvt/api/FRVT_common_2.0_draft.pdf. This includes rules for participation, hardware and operating system environment, software requirements, reporting, and common data structures that support the APIs.

1.3. Reporting

For all algorithms that complete the evaluation, NIST will provide performance results back to the participating organizations. NIST may additionally report and share results with partner government agencies and interested parties, and in workshops, conferences, conference papers, presentations and technical reports.

Important: NIST will publish the name of the developer's organization, the algorithm identifiers, and the performance results with attribution to the developer. Results will be machine generated (i.e. scripted) and will include timing, accuracy and other performance results. These will be provided alongside results from other implementations. Results will be expanded and modified as additional implementations are tested, and as analyses are implemented. Results may be regenerated on-the-fly, usually whenever additional implementations complete testing, or when new analyses are added.

Note: Due to data sensitivities, NIST does not intend on disclosing and describing the presentation attack instruments used in our evaluation. We will report PA detection metrics for each PAI but without the name or description of the PAI. This is intended to encourage broad PAD effectiveness across unknown PAs, and to discourage tuning to specific attacks. This reflects the operational reality that attackers don't advertise their methods.

1.4. Accuracy metrics

This test will evaluate algorithmic ability to detect whether an image or a video contains a presentation attack or not. Per established metrics[5] for assessment of presentation attacks, NIST will compute and report:

- Attack Presentation Classification Error Rate (APCER) – the proportion of presentation attack samples incorrectly classified as bona fide presentation
- Bona Fide Presentation Classification Error Rate (BPCER) – the proportion of bona fide samples incorrectly classified as presentation attack samples
- Attack Presentation Non-Response Rate (APNRR[6]) and Bona Fide Presentation Non-Response Rate (BPNRR) – the proportion of presentation attack and bona fide samples, respectively, that do not generate a response and fail to be processed by the algorithm software, whether it's elective refusal to process the imagery or an involuntary error. Failure to process events will be logged when the algorithm software returns a non-successful return code from the PAD function, indicating that something went wrong while processing the imagery for PAD.

We intend on reporting the above quantities for various presentation attack species.

We intend on incorporating failure to process events into the calculation of BPCER and APCER. All occurrences of failure to process by an algorithm will be treated as if a presentation attack is detected with the confidence score set to +1.

We will also publish error tradeoff plots (BPCER vs. APCER, parametric on threshold) and other analyses as appropriate.

---

[5] International Organization for Standardization: Information Technology – Biometric presentation attack detection – Part 3: Testing and reporting. ISO/IEC FDIS 30107-3:2017, JTC 1/SC 37, Geneva, Switzerland, 2017

[6] ISO/IEC 30107-3 includes APNRR "proportion of attack presentations using the same PAI species that cause no response at the PAD subsystem or data capture subsystem" to quantify outcomes where the sensor is not even triggered by the presented PA sample. We use APNRR to quantify that for the PAD-subsystem, which here is the algorithm under test.

1.5. Time limits

The elemental functions of the implementations shall execute under the time constraints of Table 2. These time limits apply to the function call invocations defined in Section 5. Assuming the times are random variables, NIST cannot regulate the maximum value, so the time limits are median values. This means that the median of all operations should take less than the identified duration. NIST will publish duration statistics.

The time limits apply per image.

Table 2 – Processing time limits in milliseconds, per 1280 x 960 image

Function	
detectPA ()	5000 (1 core)

2. Rules for participation

2.1. Participation agreement

A participant must properly follow, complete, and submit the FRVT Participation Agreement. This must be done once, either prior or in conjunction with the very first algorithm submission. It is not necessary to do this for each submitted implementation thereafter. **Note:** Organizations that have already submitted a participation agreement for FRVT Ongoing 1:1 do not need to send in a new participation agreement unless the organization updates their cryptographic signing key.

2.2. Validation

All participants must run their software through the provided FRVT PAD validation package prior to submission. The validation package will be made available at https://github.com/usnistgov/frvt. The purpose of validation is to ensure consistent algorithm output between the participant's execution and NIST's execution. Our validation set is not intended to provide training or test data.

3. Data structures supporting the API

The data structures supporting this API are documented in the FRVT - General Evaluation Specifications document available at – https://pages.nist.gov/frvt/api/FRVT_common_2.0_draft.pdf with corresponding header file named *frvt_structs.h* published at https://github.com/usnistgov/frvt/blob/pad/common/src/include/frvt_structs.h.

4. Implementation Library Filename

The core library shall be named as libfrvt_pad_<***provider***>_<***sequence***>.so, with

- provider: single word, non-infringing name of the main provider. Example: acme
- sequence: a three digit decimal identifier to start at 000 and incremented by 1 every time a library is sent to NIST. Example: 007

Example core library names: *libfrvt_pad_acme_000.so, libfrvt_pad_mycompany_006.so.*
Important: Public results will be attributed with the provider name and the 3-digit sequence number in the submitted library name.

5. API specification

Please note that included with the FRVT PAD validation package (available at https://github.com/usnistgov/frvt) will be a "null" implementation of this API. The null implementation has no real functionality but demonstrates mechanically how one could go about implementing this API.

5.1. Header File

The prototypes from this document will be written to a file named **frvt_pad.h** and are be available to implementers at https://github.com/usnistgov/frvt/blob/pad/pad/src/include/frvt_pad.h.

5.2. Namespace

All supporting data structures will be declared in the `FRVT` namespace. All API interfaces/function calls for this track will be declared in the `FRVT_PAD` namespace.

5.3. API

5.3.1. Interface

The software under test **must** implement the interface `Interface` by subclassing this class and implementing each method specified therein.

	C++ code fragment	Remarks
1.	`Class PADInterface`	
2.	`{` `public:`	
3.	`static std::shared_ptr<Interface> getImplementation();`	Factory method to return a managed pointer to the `Interface` object. This function is implemented by the submitted library and must return a managed pointer to the `Interface` object.
4.	`// Other functions to implement`	
5.	`};`	

There is one class (static) method declared in `Interface.getImplementation()` which must also be implemented. This method returns a shared pointer to the object of the interface type, an instantiation of the implementation class. A typical implementation of this method is also shown below as an example.

	C++ code fragment	Remarks

	`#include <frvt_pad.h>`
	
	`using namespace FRVT_PAD;`
	
	`NullImpl:: NullImpl () { }`
	
	`NullImpl::~ NullImpl () { }`
	
	`std::shared_ptr<Interface>`
	`Interface::getImplementation()`
	`{`
	`    return std::make_shared<NullImpl>();`
	`}`
	`// Other implemented functions`	

5.3.2. Initialization

Before any presentation attack detection calls are made, the NIST test harness will call the initialization function of Table 3. This function will be called BEFORE any calls to `fork()`[7] are made. This function <u>must</u> be implemented.

Table 3 – Initialization

Prototype	ReturnStatus initialize(	
	const std::string &configDir);	Input
Description	This function initializes the implementation under test. It will be called by the NIST application before any calls to the presentation attack detection functions of this API. The implementation under test should set all parameters.	

[7] http://man7.org/linux/man-pages/man2/fork.2.html

	This function will be called N=1 times by the NIST application, prior to parallelizing M >= 1 calls to any other functions via `fork()`. This function will be called from a single process/thread.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
Output Parameters	None	
Return Value	See General Evaluation Specifications document for all valid return code values. This function <u>must</u> be implemented.	

5.3.3. Presentation Attack Detection

The function of Table 4 evaluates presentation attack detection on still photos and/or sequential video frames. A single image or a sequence of video frames is provided to the function for detection of a presentation attack. Both PA imagery and non-PA (bona fide) imagery will be used, which will support measurement of attack presentation classification error rate (APCER) with a bona fide classification error rate (BPCER). This function <u>must</u> be implemented.

Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.

Table 4 – Presentation Attack Detection

Prototypes	ReturnStatus detectPA(	
	const Media &suspectedPA,	Input
	bool &isPA,	Output
	double &score);	Output
Description	This function takes a piece of input media containing an image or sequence of video frames and outputs a binary decision on whether the media represents a PA and a "padiness" score on [-1, 1] representing how confident the algorithm is that the piece of media contains a PA. A value of -1 means certainty that the media does not contain a PA, and +1 represents certainty that the media does contain a PA. A value near 0 will indicate uncertainty. **Question to developers:** Should this function call include an additional input parameter specifying PA intent? PAD implementations are often fielded in applications where the classes of risk are known. For example, in authentication, a primary concern is impersonation. In background-checks, the concern is of evasion, concealment. We could add a parameter that communicates "PA intent", for example: A value of 1 indicates the sample is an evasion PA or bona fide; a value of 2 indicates the sample is an impersonation PA or bona fide; a value of 3 would mean the software is not provided with that information.	
Input Parameters	suspectedPA	Input media of a single still image, or a sequence of video frames
Output Parameters	isPA	True if media contains a PA; False otherwise
	score	A real-valued score on [-1, 1] Developers are cautioned that if software only ever reports a few discrete values, the resulting error-tradeoff characteristic will have steps, such that end-users will not be able to set thresholds that finely target some objective (e.g. BPCER = 0.01). This limitation will not occur if the algorithm emits scores with a continuous distribution.
Return Value	See General Evaluation Specifications document for all valid return code values. This function <u>must</u> be implemented.	