A DISTRIBUTED MULTIMODAL BIOMETRIC AUTHENTICATION FRAMEWORK

Jonas Richiardi, Andrzej Drygajlo, Alicia Palacios-Venin,
Razvan Ludvig, Olivier Genton, Lianick Houmgny

Perceptual Artificial Intelligence Laboratory
Signal Processing Institute
Swiss Federal Institute of Technology Lausanne
`<jonas DOT richiardi AT epfl.ch>`

ABSTRACT

The design and implementation of a distributed multimodal biometric verification system inspired by the BioAPI specification is exposed. Server and Client objects divide processing responsibilities in classes linked by the Java Remote Method Invocation framework. Our framework is secured up to the core security requirements of the X9.84 standard. It is designed to be extensible to other modalities than speech and on-line signature through object-oriented design.

1. INTRODUCTION

As biometric matchers have matured significantly in the past few years and their performance has been improved substantially for many modalities, there is now a need to explore practical implementations and establish the bases of software engineering practices and designs which will serve for integration of biometric software modules. This somewhat imprecise term can take several meanings.

One is that the biometric module is a stand-alone executable, which is just given a list of data files for training and testing, and produces matching scores as an output. This is often the case in competitions; for example during the Signature Verification Competition 2004 [1] the participants were asked to submit binaries conforming to a certain command-line syntax for training and testing.

Another interpretation refers to executables that perform part of the whole biometric processing chain (feature extractors, modelling engines, matching engines,...). An example is the `HList` program, part of the `HTK` toolkit [2], which can be used to extract features from speech files.

Yet another interpretation refers to generic pattern recognition/machine learning source code or toolkits in a particular programming language that can be used and applied to the biometric case. Examples abound, from C++ based toolkits such as ALIZE [3] to the Matlab Netlab library [4].

Standardisation efforts have matured largely in the industrial context, with the appearance of industry standards such as BioAPI [5, 6] and X9.84 [7], and more effort under way at the ISO level (ISO/IEC JRC 1/SC 37 series). Because we feel that most biometric researchers are rightly more interested in focussing their efforts on improving classifiers and combination methods than in following industry standards, we propose a flexible framework (the Multimodal Biometric Authentication Framework or MBAF) which can be used to experiment with biometric authentication in a distributed, secure system. It offers a large amount of flexibility in integration of existing biometric software modules and provides remoting, persistence, security, GUI, and workflow services. We believe this to be the first freely available distributed object framework of the kind, and it was used at the BioSecure residential workshop in Paris in August 2005.

In the rest of the paper, we talk about distributed biometric verification architectures (Section 2), summarize industry standards that are applicable to our framework (Section 3), describe the design and implementation of the framework (Section 4), show the principles of integration of external code into the MBAF (Section 5), and finally draw some conclusions and outline future work in Section 6.

2. DISTRIBUTED BIOMETRIC AUTHENTICATION

Many architectures exist for biometric verification systems. In abstract, functional terms, they mostly differ by how the processing steps for biometric data are divided between different machines. Here, we present a succint description of processing steps which are common to many biometric modalities.

Capture or acquisition The biometric data (voice, on-line signature, fingerprint, . . .), also called biometric presentation, is digitised via the input device (microphone, pen tablet, fingerprint scanner, . . .) and stored in memory.

Preprocessing The signal-domain acquired data is prepared for feature extraction. This is typically used for normalising the signal-domain data and remove biases or sources of corruption in a systematic fashion.

Feature extraction Discriminative features are extracted from the preprocessed data. Although features are very different for each biometric modality, the general underlying principle remains the same: this processing step typically reduces the dimensionality of the input data to create a feature-level representation of input patterns that will be used by the classifier to perform pattern recognition.

Postprocessing Features are normalised to remove bias or adapt them to the classifier.

Template creation User models, also called templates, are created from training feature sets to obtain a generic representation of a user that will be used for future comparisons. Many algorithms and can be used depending on modality/features and model class, with some "models" corresponding directly to features.

Background model creation A background model, also called world model or anti-model, is needed by some biometric algorithms to provide normalisation for user presentation scores. They represent an "average" of the users from the population of the system.

Template storage Once their parameters are estimated, user models are stored in a secure location for use in later biometric operations.

Template matching A biometric presentation is compared with a particular user's biometric template. This typically results in a presentation score which is somehow related to how likely it is that the particular user is the source of that presentation. Depending on model and classifier types, this processing step will vary.

Threshold computation Several presentations belonging to a particular user and several presentations not belonging to that particular user (impostor presentations) are matched to that user's model to determine a hard limit (the threshold) below which a presentation will not be considered as belonging to the user.

The processing steps described above will be used in the following higher-level biometric *operations*, represented as an activity diagram in Figure 1.

Enrolment A user is added to the biometric system.

Verification The claim to a user's identity causes the presented biometric data to be compared against the claimed user's model.

Identification A database of user models is searched for the N most likely sources (N-best list) of the biometric presentation.

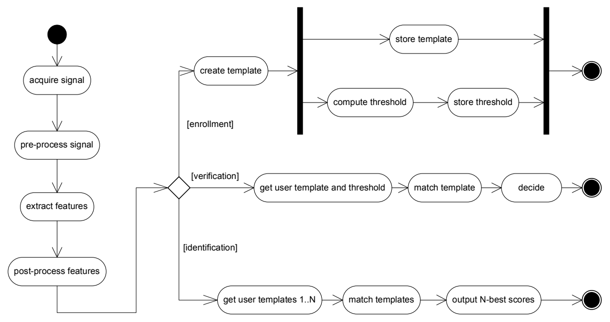


Figure 1: *biometric operations and corresponding processing steps for a generic modality*

Biometric system architectures mostly differ by how the steps (and hence operations) above are divided between different machines. In *off-line* systems, the access point has enough computing power and is trusted sufficiently to perform all the steps above locally. In *on-line* systems, the acquisition, pre-processing, and feature extraction are typically done on one of numerous client access points, and template creation, storage, and verification is performed on a biometric verification server.

Our framework defers all front-end tasks to the client, thus meaning that all pre-processing, feature extraction, and post-processing of feature is done locally and that only features travel through the wire. This offers significant advantages in terms of network bandwidth usage reduction and associated reduction of encryption overhead (see Section 2.3). It is however assumed that the client will have sufficient processing power to run a Java virtual machine and perform the needed operations, and have properly configured acquisition devices (microphone, pen tablet, fingerprint scanner...).

2.1. Middleware for distributed biometric applications

In previous work, we have higlighted that packet loss on IP networks would be detrimental for biometric verification over the internet, and advocated against the use of UDP for biometric data transmission. Furthermore, we have pointed out that packet reordering, a prevalent issue on the Internet, could also be damaging depending on the model and hence recommended using TCP [8].

Here we push this idea further by advocating for the use of middleware in our distributed biometric verification system, rather than doing a grounds-up implementation using socket programming and reinventing the wheel.

Object-oriented middleware allows objects on one host in a network to invoke a method on an object located on a different host. Typically, the middleware will handle at least data marshalling and unmarshalling, data types heterogeneity, and remote/local object references. Many choices are suitable for a distributed biometric verification system, for instance CORBA, COM, or Java RMI.

2.2. Java as middleware in biometric systems

In the presented framework we use Java RMI because it offers platform independence for both client and server objects and is less complex than CORBA to implement and deploy. Furthermore, the Java API offers many useful functionalities such as sound, cryptography and security functions and database access out of the box. Java is also flexible in offering binding mechanisms for native methods (the Java Native Interface or JNI) written in other languages, which means some parts of the system can be implemented in fast C++ code.

Lastly, if a three-tier architecture became necessary, for instance in the case of e-commerce transactions protected by biometric authentication, Java would offer an easy transition in the form of a Java Server Pages, applets, and web services combination.

2.3. Distributed verification security

Distributed biometric verification systems must contend with many challenges if they are to be made secure. Many of these challenges are shared by other distributed applications not related to biometric verification, and there is a large body of knowledge available to draw from in different fields [9].

In essence, biometric and application data should not be acquired or tampered with by third parties while being transmitted over the network (*leakage*), only authenticated clients should connect to authenticated servers, the systems should be able to cope with denial of service attacks, etc. Of critial importance is also the safeguarding of biometric templates and enrollment/adaptation data stored in the biometric database.

In this respect, the safest possible architecture is one where biometric templates or enrollment data are not stored in a central location, but instead stored on a smartcard-type device that each user carries around. The more biometric data a central repository holds, the more its value will make fraudulent acquisition or tampering likely. The current system has no support for smartcard (distributed) template storage.

Java supports certificates, symmetric and asymmetric encryption, secure socket layer (SSL) connections and more security features through the Java Authentication and Authorization Service (JAAS) framework, the Java Cryptography Architecture framework, the Java Cryptography Extension (JCE), and the Java Secure Socket Extension (JSSE). Our framework uses both client and server X.509 certificates to authenticate both ends of the transaction. It also encrypts all communications between the client and server by using an SSL tunnel. Lastly, all biometric data stored on the database is encrypted using a symmetric encryption scheme, and decrypted on-the-fly as needed. We also use JAAS to implement user roles, so that administrators can enroll new users, but users can only verify themselves.

3. APPLICABLE INDUSTRY STANDARDS

3.1. The BioAPI specifiction

The BioAPI specification 1.1 (ANSI/INCITS 358-2002 standard) specifies data types, module functionalties, function names, data storage specifications and other items to promote interoperability of biometric technologies from different vendors and with different modalities. It defines high-level abstractions such as `Enroll` or `Verify`, which in turn can rely on primitive functions such as `Capture`, `Process`, `Match`, and `CreateTemplate`; these are assigned specific signatures and functionalities. The specification allows for much flexibility in the distribution of processing steps, and supports both off-line and on-line systems.

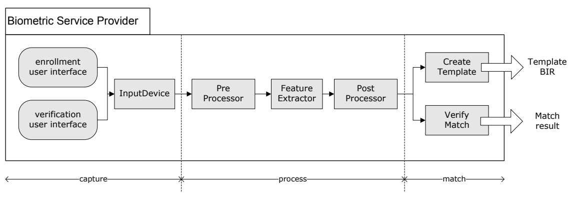


Figure 2: *Biometric Service Provider implementation*

One of the main concepts in BioAPI 1.1 is the Biometric Service Provider (BSP). As shown in Fig. 2, a BSP is responsible, for a given modality, to perform all the steps described in Section 2. Essentially, the BioAPI specification provides details of hooks into, and output data from, BSPs.

More recently, a new BioAPI specification (BioAPI v2.0) has been published [6]. It extends and generalises the BioAPI 1.1 framework. An interesting change is that it breaks down BSPs into sub-units called Biometric Function Providers (BFPs). To these functional level correspond "unit" levels, which replace the concept of "device" found in BioAPI 1.1. Thus, it is now possible for a BioAPI 2.0 BSP to use several sensors (sensor units).

The presented system uses an architecture that is inspired by the BioAPI specification in terms of object design and functional units, but is implemented in Java, and has no registered format owner for the biometric data. Thus, it is not BioAPI compliant. If needed, the system could be modified to make use of BioAPI-compliant BSPs by using the Java-to-BioAPI BSP bridge provided by Gens Software.

3.2. The ANSI X9.84 standard

The ANSI X9.84 standard [7], *Biometric Information Management and Security for the Financial Services Industry*, describes the security features needed to implement biometric verification for financial services. The Core Security Requirements are summarised here:

1. The integrity of biometric data and verification results must be guaranteed between any two components using software techniques such as hashes and physical measures such as tamper-resistent assemblies.
2. The source and receiver of biometric data and verification results must be authenticated, again using software or physical methods where appropriate.
3. The confidentiality of biometric data may be ensured between any 2 components.

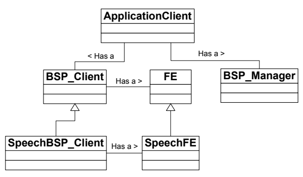
X9.84 also describes secure enrollment, verification, storage, transmission, and termination procedures.

Assuming the physical security of the machines our software, the security features exposed in section 2.3 make our framework compliant with the Core Security Requirements of X9.84.

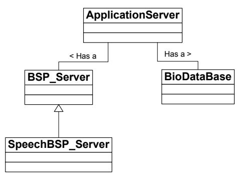
4. FRAMEWORK DESIGN AND IMPLEMENTATION

4.1. Object hierarchy

The partial class diagram of Fig. 3 presents a simplified overview of the MBAF, showing only class derivations for the speech modality and hiding member data and methods.



(a) Client-side objects



(b) Server-side objects

Figure 3: Partial class diagram with client-side and server-side components

The application uses three main objects: `BSP_Client`, which is intended to run on the access points such as desktop PCs, laptops, PDAs, ATMs, or other platform with IP network connectivity, Java support, and sufficient processing power. It is responsible for data acquisition, preprocessing, feature extraction, and user interface management. `BSP_Server`, which runs on the biometric verification server, typically a server PC running some flavour of Windows or Linux. This object is responsible for creating user templates, storing them in the biometric database, and scoring biometric presentations with respect to a particular user's template. `BSP_Manager` is responsible for loading and unloading of BSPs (modalities).

The `BSP_Client` is a base class that has one main class to perform front-end duties, `FE`. The latter has five generic classes, each responsible for an aspect of biometric data processing, matching the generic steps presented in Section 2. These are `InputDevice`, which can be specialised depending on the acquisition hardware (for the speech modality, `Microphone` is derived from `InputDevice`, and allows to set sampling rate, quantisation, etc.), `Preprocessor`, which can be specialised according to the modality and device being used, FeatureExtractor, and Postprocessor. Each new modality added to the system can be implemented in Java using classes derived from the five generic classes above.

Similarly, on the server side, SpeechBSP_Server is derived from the BSP_Server base class because the BioSPI_VerifyMatch and BioSPI_CreateTemplate methods have some specificity for each modality, though in the case of speech or signature modelling the same model training engine (expectation-maximisation for learning Gaussian mixtures) could be used.

This object design is sufficiently generic to meet the needs of many other verification modalities such as fingerprints or face.

4.2. Biometric data storage

Biometric templates need to be stored in a database for the BioSPI_VerifyMatch to function. The framework provides format-agnostic features and model storage, retrieval, and deletion services. This is accomplished using the *MySQL* open-source database, to which the Java code connects using an open-source type IV Java Database Connectivity driver, Connector-J.

5. EXTERNAL MODULES AND NATIVE CODE

The framework uses the Java Native Interface (JNI) to be able to call methods on objects implemented in C++. The way external code is currently integrated into our framework is that JNI method signatures with JNI datatypes are added at global scope in the C/C++ source code of the biometric engine, the engine is recompiled into a shared object under Linux or a dynamic link library under Windows, and a minimal wrapper is written in Java to call on the native library. Biometric features and models are exchanged between the Java code and the native code by using the filesystem available on the server. While this may not be the most elegant solution, it offers the most flexibility for adding new biometric modules to our framework.

Work is currently underway to provide more options to integrate native code at different levels, for instance it might be interesting to use a native feature extractor somewhere in the processing chain. Also, a command-line wrapper structure will be added for the case where source code is not available.

5.1. Native library: Speech modality

The Torch C++ library [10] is used for training Gaussian Mixture speaker models and scoring speech presentations, and a JNI interface is used to bind to the SpeechBSP_Server Java object. File lists and feature files are simply put on disk and fed to the Torch engine.

On the client side, speech preprocessing consists of DC bias removal and silence removal. The features extracted are 12 MFCCs, deltas and delta-deltas. On the server side, speakers are modelled using a 64 components Gaussian Mixture models with diagonal covariance matrices World model normalisation is used.

5.2. Native library: On-line signature modality

The ALIZE C++ toolkit [3] is used to for training GMMs for signature modality users and scoring signature presentations. A similar JNI interface to that used for Torch links the native shared library to the Java framework.

The signature verification process is similar to that described in [11].

6. CONCLUSIONS

We have presented the design and implementation of a distributed framework for multimodal biometric authentication. The framework offers network transparency, end-to-end encryption, and biometric data management services as well as a skeleton for biometric authentication applications in the form of a set of base classes in Java which can be derived and extended.

Future work include making the integration of native code more flexible and with as little rewrite as possible (possibly none), making parallel modality use possible, and translating and improving the documentation and user manual.

The latest version of the MBAF with at least example speech and signature modalities can be downloaded from http://scgwww.epfl.ch/jonas/software.html.

7. ACKNOWLEDGEMENT

Part of this work was funded by a European Commission grant as part of the BioSecure network of excellence.

8. REFERENCES

[1] D.-Y. Yeung, H. Chang, Y. Xiong, S. George, R. Kashi, T. Matsumoto, and G. Rigoll, “SVC2004: First international signature verification competition,” in *Proceedings 2004 Biometric Authentication: First International Conference, (ICBA 2004)*, (Hong Kong, China), pp. 16–22, July 2004.

[2] S. Young, G. Evermann, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, “The HTK book,” tech. rep., Cambridge University Engineering Department. Available at http://htk.eng.cam.ac.uk.

[3] J.-F. Bonastre, F. Wils, and S. Meignier, “ALIZE, a free toolkit for speaker recognition,” in *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2005)*, (Philadelphia, USA), pp. 737–740, March 2005.

[4] I. Nabney, *Netlab: Algorithms for pattern recognition*. Advances in pattern recognition, Springer, 2004.

[5] American National Standards Institute, *Information technology - BioAPI Specification (Version 1.1) (formerly ANSI INCITS 358-2002)*. New York, USA: American National Standards Institute, 2002.

[6] International Standards Organisation, *ISO/IEC 19794-1:2005 (BioAPI 2.0)*. Geneva, Switzerland: International Standards Organisation, 2005.

[7] American National Standards Institute, *Biometric Information Management and Security for the Financial Services Industry: ANSI X9.84-2003*. New York, USA: American National Standards Institute, 2003.

[8] J. Richiardi, J. Fierrez-Aguilar, J. Ortega-Garcia, and A. Drygajlo, “On-line signature verification resilience to packet loss in ip networks,” in *Proc. 2nd COST 275 Workshop on Biometrics on the Internet: fundamentals, advances and applications*, (Vigo, Spain), pp. 9–14, March 2004.

[9] R. Anderson, *Security engineering: a guide to building dependable distributed systems*. John Wiley and Sons Inc., 2001.

[10] R. Collobert, S. Bengio, and J. Mariéthoz, “Torch: a modular machine learning software library,” Technical Report IDIAP-RR 02-46, IDIAP, 2002.

[11] J. Richiardi and A. Drygajlo, “Gaussian mixture models for on-line signature verification,” in *Proc. ACM Multimedia 2003 Workshop on Biometric Methods and Applications*, (Berkeley, USA), Nov. 2003.