Chapter 14
A Framework for Secure Selfie-Based Biometric Authentication in the Cloud

Veeru Talreja, Terry Ferrett, Matthew C. Valenti and Arun Ross

Abstract Cloud-based selfie authentication has multiple advantages over on-device selfie authentication: Cloud-based authentication can support nomadic access from multiple devices including those not owned by the user, can leverage cheap and scalable utility computing, and can enable rapid innovation by allowing new matching algorithms to be continually deployed with no need to update the local device. This chapter presents a framework for a cloud-based selfie biometric authentication, which is termed *Selfie-Biometrics-as-a-Service* (SBaaS). By leveraging Platform-as-a-Service (PaaS) concepts, the framework is designed to enable independent software vendors to develop extensions and add-ons to a provider's core application. In particular, the framework creates an innovative marketplace for biometric algorithms by providing a standard pre-built interface for the development and submission of new matching algorithms. When an authentication request is submitted, a criteria is used to select an appropriate matching algorithm. Every time a particular algorithm is selected, the corresponding developer is rendered a micropayment. Also presented in this chapter are solutions for preserving the confidentiality of biometrics stored in the cloud. This can be achieved through the use of biocryptosystems, which are secure biometric architectures involving the conversion of biometric features into secure signals that can be stored in the biometric database but are still useable for authentication. To provide a concrete example, a case study of a selfie-based ocular recognition system is disclosed, and detailed descriptions are provided of the user and developer interfaces.

V. Talreja · T. Ferrett · M. C. Valenti
West Virginia University, Morgantown, WV, USA
e-mail: vtalreja@mix.wvu.edu

T. Ferrett
e-mail: terry.ferrett@mail.wvu.edu

M. C. Valenti
e-mail: matthew.valenti@mail.wvu.edu

A. Ross (✉)
Michigan State University, East Lansing, MI, USA
e-mail: rossarun@cse.msu.edu

A. Rattani et al. (eds.), *Selfie Biometrics*, Advances in Computer Vision and Pattern Recognition, https://doi.org/10.1007/978-3-030-26972-2_14

14.1 Introduction

Recently, there has been tremendous interest in incorporating selfie biometric solutions into consumer electronic such as smartphones and tablets. Traditionally, specialized biometric sensors have been proprietary with high cost, low market adoption, and limited compatibility with competing systems [28]. However, smartphones are equipped with cameras and other sensors suitable for biometric sensing tasks in the context of face, fingerprint, ocular, and gait recognition. The presence of high-resolution front-facing cameras, in particular, offers the possibility of performing selfie biometric recognition within the confines of the device.

Selfie biometric authentication is currently being used by e-commerce companies to enable customers to purchase merchandise more conveniently, by using a selfie for login authentication and payment confirmation. It is also being used by banks and other organizations to protect the security of financial records and access to funds. Major organizations that have begun using a selfie authentication technology include Amazon, Mastercard, and Alibaba.

When selfie biometrics are used to authenticate a user on a personal device, the enrollment is usually local to the device and successful authentication unlocks a locally stored key for use in a conventional private-key cryptosystem. However, performing selfie biometric authentication using only the resources local to the device is challenging due to several confounding factors including variations in head pose, ambient illumination, facial expression, occlusion, and limited availability of resources within the smartphone for storage and computation [11, 13]. Another significant challenge that cannot be met by a solitary personal device is supporting the nomadic usage habits demanded by today's consumers, who want to be able to gain access to services from any location using any number of personal devices or from public infrastructure such as ATMs and pay stations. As an alternative to confining authentication to the local device, biometric authentication can be performed in the cloud. In this regard, *cloud computing* may be harnessed as a viable option. Cloud computing [19] facilitates the outsourcing of computing and storage tasks to infrastructures managed by dedicated providers—providing an opportunity to surpass mobile resource limits [29]. For instance, the feature extraction, data storage, and matching components of a selfie biometric system can be moved to a cloud infrastructure, while leaving only the sensing task in the device. This is an example of a *biometrics-in-the-cloud* paradigm [10].

There is an increased interest in performing biometric recognition in mobile devices and as a cloud-based service [2, 4, 9, 12, 16]. In [5], a framework for cloud-based face recognition emphasizing the parallelization of recognition tasks across multiple servers is introduced. Performing biometric recognition in mobile devices and as a cloud-based service has also been adopted widely in the biometric recognition industry. For example, Zoloz provides cloud-based selfie biometric authentication solutions, which are used by around 50 banks and 200 million users in Asia. FacePhi offers a cloud-based mobile facial recognition solution, Selphi, that enables mobile banking users to access their accounts just by taking a selfie.

There are a variety of models for providing biometrics in the cloud. *Biometrics-as-a-Service* (BaaS) is a model where the biometrics-in-the-cloud architecture is offered by a service provider [28]. If the infrastructure allows for component developers to develop and incorporate custom components in the cloud (e.g., feature extraction or matcher modules), then it is referred to as *Platform-as-a-Service* (PaaS) [18]. Broadly speaking, some PaaS providers, such as Bungee Labs and SalesForce.com, provide a framework that allows independent software vendors (ISV) to develop extensions or add-ons to the provider's core application [3]. A key contribution of this chapter is to present a similar framework for cloud-based selfie biometric authentication known as *Selfie-Biometrics-as-a-Service* (SBaaS) that allows the developers of biometric recognition algorithms to actively contribute to the SBaaS system. This is achieved by creating an interface for uploading algorithms and a scheme for selecting algorithms and rendering micropayments to the developers. Having such an infrastructure in place has the benefit of promoting innovation and reducing costs for the BaaS by allowing the development of its key components to be outsourced [32].

Authentication in the cloud may raise questions regarding the preservation of information confidentiality and the use of secure authentication methods in the cloud. Privacy and security of the biometric data can be achieved by combining cloud authentication modules with biometric security architectures involving the conversion of biometric features into secure signals that can be stored in the biometric database but are still usable for authentication.

In this chapter, we present a cloud-based framework SBaaS for performing *selfie biometric recognition* using smartphones or other mobile devices as sensors and demonstrate a reference implementation of this framework using ocular recognition as a specific example. The salient features of this framework include the following:

1. Smartphones and other mobile devices, including the cameras resident in them, require no modification from their stock hardware configuration.
2. Computationally intensive tasks such as segmentation, feature extraction, and matching are outsourced to the cloud.
3. Software developers can upload their own biometric matching algorithms to the cloud. Thus, the cloud hosts multiple matching algorithms pertaining to multiple developers.
4. Enabling developers to upload matching algorithms creates an environment where the value of algorithms is measured by their in-application performance, creating incentives for competition and innovation.
5. The matching algorithm is automatically selected based on the characteristics of the input images and the performance of the algorithm as evaluated on sequestered data.
6. Every time an algorithm is selected for matching, its developer is credited under a *micropayment* model.

In addition to the SBaaS framework, we also present a *secure* Selfie-Biometrics-as-a-Service (SSBaaS) framework, in which the biometric features are converted into secure signals using secure biometric architectures to preserve the privacy information of the user.

14.2 Architecture

This section develops a general framework for SBaaS, which can be implemented for any biometric modality. The components of the system are the *user interface*, *developer interface*, *biometric database*, and cloud-based *computing infrastructure*. The user interface is an application, which may be embodied as a mobile-enabled web application or a native smartphone app, through which users submit matching requests. In particular, users use the interface to submit a selfie image to be compared with a corresponding enrollment selfie image stored in the biometric database, where the biometric database is a storage location for enrollment selfies. The developer interface is a virtual machine having identical software as the computing infrastructure for developing and submitting matching algorithms to the system. The computing infrastructure consists of a cloud server for executing matching requests using developer-submitted algorithms. The architecture is depicted in Fig. 14.1.

14.2.1 Cloud Computing Characteristics

The definition of cloud computing [19] encompasses several elements that the SBaaS architecture provides. Users can submit matching requests through a web interface,

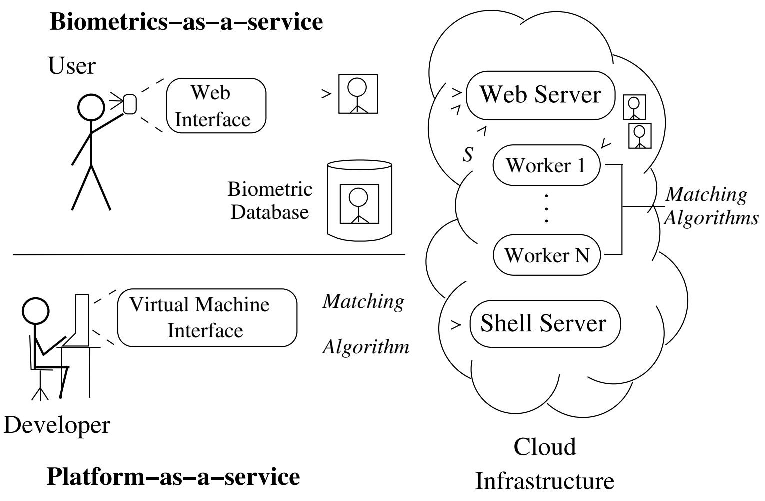


Fig. 14.1 Proposed SBaaS architecture. A user submits a matching request by uploading a probe selfie captured using the camera on a mobile device. The probe selfie from the user and the corresponding enrollment selfie from the biometric database are submitted to the cloud infrastructure. The comparison is performed by a worker process. Matching algorithm developers use the virtual machine interface to develop and submit their algorithms to the cloud infrastructure over a shell session for deployment to the worker processes

and the requests are automatically processed by available servers, providing *on-demand self-service*. The web interface is designed for both mobile and desktop use, incorporating *broad network access*. *Resource pooling* is implemented such that multiple matching requests are distributed among servers, automatically balancing the request load as needed.

Additional servers can be added to the system rapidly, as the operating system installation and software provisioning are fully automated, providing *rapid elasticity*. The computing framework software is designed to execute matching requests across any number of servers. The execution of each matching request is tracked, and users are charged for service in proportion to the number of completed requests and according to the selected algorithms (i.e., not all algorithms will be priced the same). Matching algorithm developers are credited for every matching request that uses their algorithm, making this an instance of *measured service*.

Cloud services are classified according to the level of abstraction at which the users and developers interact with the infrastructure. In the SBaaS architecture, users perform matching operations by submitting requests through a web interface. In the context of biometrics, this architecture is an instance of *biometrics-as-a-service*, a model for providing biometric recognition functionality through a service provider [28]. A virtual machine containing an operating system and pre-installed software that is identical to the cloud infrastructure is provided to the software developers, making this an instance of *Platform-as-a-Service*. The use of a virtual machine ensures that developed algorithms are binary compatible with the infrastructure and obviates the need for developers to provision their own development environments.

14.2.2 User Interface

The *user interface* shown in Fig. 14.1 is a web application that is accessible on both mobile and desktop browsers. This interface is used by the user to submit a probe selfie for matching. The probe selfie and the enrollment selfie from the biometric database are sent to the cloud infrastructure where preprocessing is done to select the most suitable matching algorithm. The matching algorithm can be selected automatically by the system, or the user can also select the matching algorithm through the user interface. The selected algorithm is executed on the selfies and a match score is returned. Even though Fig. 14.1 shows that the user interface is embodied as a web application, it could also be implemented as a native app that runs directly on the smartphone. While it could be a stand-alone app, it could also be integrated into another app, such as a banking or e-commerce app.

14.2.3 Biometric Database

The *biometric database* stores the enrollment selfies of the users. When a user submits a matching request through the user interface by uploading their selfie to the cloud, the corresponding enrollment selfie from the biometric database is also uploaded to the cloud infrastructure for authentication. The biometric database may be centralized, or it may be distributed. For instance, the biometric database could be stored on the user's smartphone as part of the image gallery or it could also be a part of the cloud infrastructure. Another example of a biometric database is that the user can store their enrollment selfie in their personal accounts with a storage provider such as Dropbox, which would cater to the needs of a nomadic user for access from multiple devices. Alternatively, the biometric database could be held by an entity such as a bank or an e-commerce site, or it could also be hosted by a third-party authentication service provider. This idea of third-party authentication service provider is analogous to the password authentication service rendered by, for example, Facebook or Google.

14.2.4 Computing Infrastructure

The computing infrastructure executes matching requests using developer-submitted algorithms. The user submits a matching request to the web server by uploading a probe selfie and, optionally, selecting a matching algorithm. The other selfie for the matching request is the enrollment selfie from the biometric database. If the user does not select an algorithm, the system selects one automatically. The server stores the selfies as data files in the user's *data directory* and creates a *job file* in the user's *job input queue* containing parameters required for matching, such as filesystem paths to the input files and to the matching algorithm, if an algorithm has been selected.

The *job manager* preprocesses the job file and moves it to the user's *job running queue*. Preprocessing is performed as follows. If the user selected a specific matching algorithm, no action is taken during preprocessing. If no algorithm was selected by the user, the system selects an algorithm automatically depending on the characteristics of the probe selfie. A table of matching algorithms and the number of times each has been executed is updated by the job manager, incrementing the number of executions for the selected algorithm. Following preprocessing, the job manager creates a *task file* containing the paths to the input files and the selected matching algorithm and places the task file in the *user task input queue*.

The *task controller* determines the number of tasks contained in all users' task input queues and schedules tasks for execution such that all users receive an equal share of the available processing cores, which is an instance of *fair scheduling*. The system is implemented such that other scheduling policies may be incorporated. To schedule an individual task for execution, the task controller moves the task file from the user's task input queue to the *global task input queue*.

A *generic worker process* running on one of the worker nodes reads a task in the global task input queue and moves it to the *global task running queue*. The generic worker process executes the matching operation, saves the result in the task file, and moves it to the *global task output queue*. The task controller moves the task from the global task output queue to the *user task output queue*. The task file is read by the job manager, and the matching result is stored in the job file and moved to the *job output queue*. The web interface reads the matching result from the job file and displays it to the user.

14.2.5 Developer Interface

The *developer interface* is a *virtual machine* (VM) that contains software for developing algorithms for submission to the cloud infrastructure. The operating system and software environment on the VM are configured identically to the environment on the computing infrastructure (i.e., the cloud server). Identical configuration obviates the need for the algorithm developer to invest time installing and configuring a compatible development environment. The VM enables the developer to implement matching algorithms that are binary compatible with the infrastructure, and upload scripts and executables directly for use.

The VM is distributed over the Internet as an archive containing a disk image of the preinstalled and configured operating system. The *software hypervisor*, which executes the virtual machine, is chosen for compatibility with as many widely used host operating systems as possible. A desktop environment having minimal resource requirements is chosen for the VM to enhance user interface performance in a virtualized environment.

The VM contains software for algorithm development, such as compilers, text editors, debuggers, and source control clients, as well as libraries commonly used for image processing applications and research. The developer must implement their algorithm such that it can be executed on a command line—a broad and general requirement that is straightforward to satisfy.

14.3 A Reference Infrastructure Implementation

This section describes a reference implementation of the general SBaaS architecture described in Sect. 14.2. The user interface implementation is first described, followed by the developer interface. Finally, details of the computing infrastructure implementation are given. For the implementation given in this section, the biometric database is considered to be the image gallery on the user's smartphone or tablet.

14.3.1 User Interface

The user interface is implemented using Mobile-Google Web Toolkit (MGWT)[1] which is a software framework for developing mobile web applications. MGWT is an extension of Google Web Toolkit (GWT),[2] which is a Java-based framework, for creating efficient and optimized browser-based applications. GWT is an open-source completely free framework that helps developers to build high-performance web applications without having expert skills in JavaScripting or browser quirks. Google also uses GWT in many of its products such as Inbox, Calendar, Adwords, and AdSense.

While GWT can help build fast desktop applications using Java, it lacks widgets and animations for developing mobile apps. MGWT closes this gap—MGWT provides mobile widgets, smooth animations, touch support, and much more. One can use MGWT to build highly optimized Java-based AJAX applications that are compatible with all browsers, including Android and the iPhone mobile browsers. We used MGWT 1.1.2 along with GWT 2.7 and Eclipse to develop the user interface. A few other Java-based API's were also used along with MGWT to develop the functionality of the user interface.

The steps taken by a user to submit a job through the web interface and obtain the results of the matching are as follows:

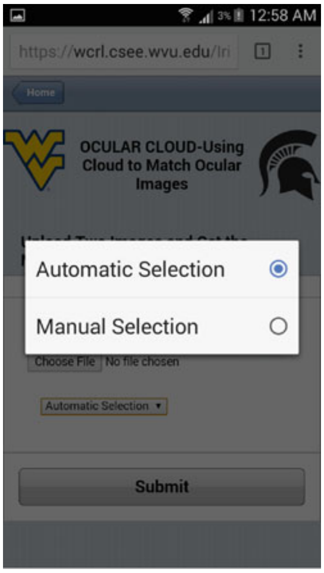
1. The web application is accessed by pointing a mobile browser to a known Web site.
2. After logging into the application, the user can either view their previous job submissions or submit a new job.
3. If the user wishes to submit a new job, they can upload a probe selfie, a gallery selfie, and either explicitly select a matching algorithm or allow the interface to select an appropriate algorithm based on image characteristics, as shown in Fig. 14.2a.
4. Upon submitting the job request, the user will be redirected to the Job History page. Shown in Fig. 14.2b is the Job History page view. This page provides details about all previous jobs submitted by the user. The user can view the complete details of a particular job—including the input images and the matching score—by clicking on the associated job.

14.3.2 Computing Infrastructure

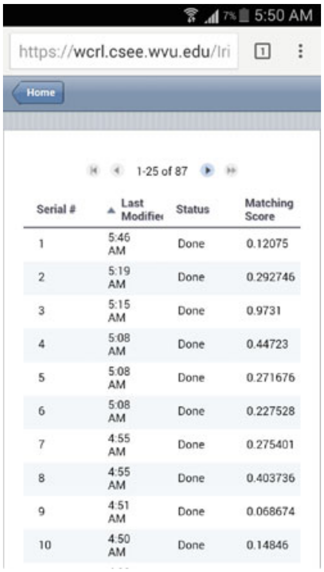
The computing infrastructure used in this reference implementation is a heterogeneous cloud of servers, each having a varying number of processing cores and main

[1]http://www.m-gwt.com/.

[2]http://www.gwtproject.org/.



(a) Algorithm Selection page



Serial #	Last Modifie	Status	Matching Score
1	5:46 AM	Done	0.12075
2	5:19 AM	Done	0.292746
3	5:15 AM	Done	0.9731
4	5:08 AM	Done	0.44723
5	5:08 AM	Done	0.271676
6	5:08 AM	Done	0.227528
7	4:55 AM	Done	0.275401
8	4:55 AM	Done	0.403736
9	4:51 AM	Done	0.068674
10	4:50 AM	Done	0.14846

(b) Job History page

Fig. 14.2 Screen shots of the mobile web app

memory. A single server acts as a router between the Internet and remaining servers and hosts the web and shell servers. This server is denoted as the *head node*. The remaining servers execute matching requests and are denoted as *worker nodes*. The operating system on all nodes is *Ubuntu Linux*. All data are stored in the head node and shared to the worker nodes using the standardized distributed file system protocol *Network File System* (NFS). The software components implementing the job manager, task controller, and generic worker are designed to work independently of one another, communicating through files on the file system. This architecture allows the components to be reused with little or no modification to the code, consistent with the UNIX philosophy and the notion of a *microservice* [35].

The job manager is implemented as a MATLAB® program that runs persistently on the head node within a *GNU screen* session. When a user submits a matching request using the web interface, the interface creates a data file (in MATLAB's *.mat* format) containing (a) the paths to the input images and (b) the user's algorithm selection option; this file is stored in the user's home directory. The job manager creates a task file—also in *.mat* format—containing paths to the images and the algorithm to execute.

Like the job manager, the task controller is implemented as a MATLAB program on the head node that is run in a GNU screen session. The task controller schedules a user's task for execution when the worker node resources become available. Exactly one matching request may be executed for every processing core available on the worker nodes. Once this limit is reached, further matching requests must wait until a core becomes available.

A matching request is executed by a generic worker process running on a worker node. The generic worker process is a MATLAB program which executes the algorithm specified in the task file. The task file specifies an *entry function* for the matching algorithm, which is a MATLAB function implemented by the algorithm developer to initiate algorithm execution. Since the algorithm developer has full control of the entry point function, they may execute a program implemented in any language which can be executed on the Linux command line by using the MATLAB feature to execute shell commands.

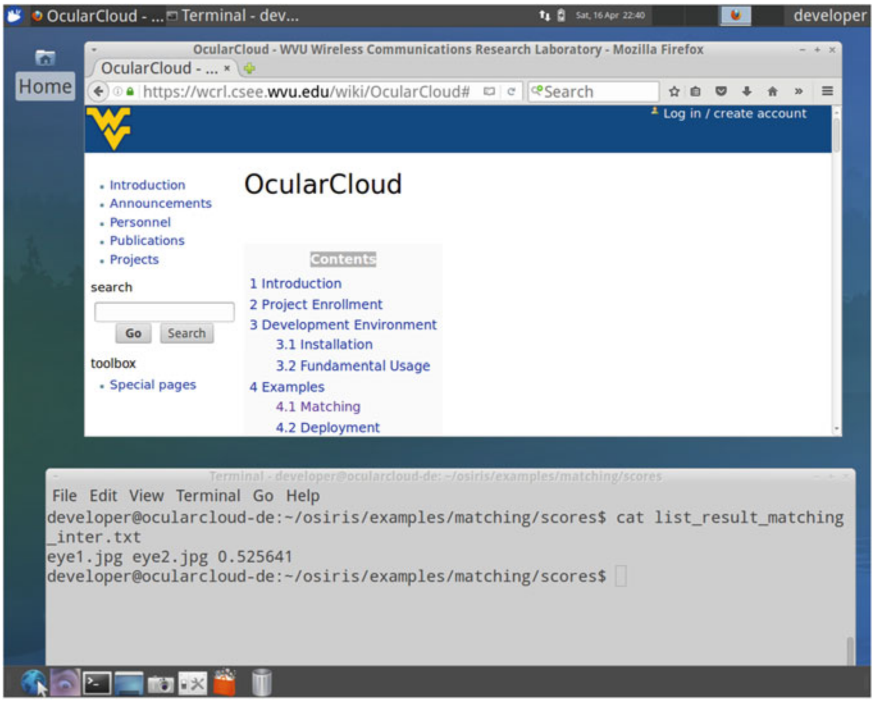
Once the algorithm execution is complete, control is returned to the generic worker, which stores matching results in the task file. The task file is consumed by the job manager, which stores the matching result in the job file. The job file is passed through the queues to the web interface, which displays the matching result to the user. In this reference implementation, the matching score is sent back to the user.

14.3.3 Developer Interface

The developer interface is implemented as a virtual machine using *Ubuntu* as the operating system. This is the same operating system installed on the cloud infrastructure nodes, which simplifies the deployment of matching algorithms. The software tools and libraries used for compiling algorithms in the developer interface exactly match those on the infrastructure, enabling the developer to deploy binaries directly. Virtualbox was chosen as the hypervisor as it is freely available for all major computing platforms (Windows, OSX, and Linux). An example of the developer interface is shown in Fig. 14.3.

The virtual machine is distributed through a publicly accessible Web site as a compressed archive, which expands to a single Virtualbox Disk Image (VDI) file. The developer specifies the VDI file as the disk image for a virtual machine in Virtualbox, and the developer interface is immediately available. Downloading and executing a virtual machine image is much simpler than a conventional provisioning process where the developer personally installs Ubuntu and the required software.

The virtual machine contains various machine learning frameworks, an open-source computer vision library Open CV *2.4.11* [7], and standard utilities for software development in the Linux environment such as GNU Emacs and the GNU compiler collection. Documentation for using the interface is provided as a wiki, which is linked via the interface desktop. The developer deploys their algorithm by uploading the required scripts, executables, and data files to their home directory in the cloud and submitting a request for integration to the infrastructure administrator.


OcularCloud - ...
Terminal - dev...
developer
Home
OcularCloud - WVU Wireless Communications Research Laboratory - Mozilla Firefox
OcularCloud - ...
https://wcrl.csee.wvu.edu/wiki/OcularCloud#
Search
Log in / create account
Introduction
Announcements
Personnel
Publications
Projects
search
Go
Search
toolbox
Special pages

OcularCloud

Contents
1 Introduction
2 Project Enrollment
3 Development Environment
3.1 Installation
3.2 Fundamental Usage
4 Examples
4.1 Matching
4.2 Deployment

Terminal - developer@ocularcloud-de: ~/osiris/examples/matching/scores
File Edit View Terminal Go Help

```
developer@ocularcloud-de:~/osiris/examples/matching/scores$ cat list_result_matching
_inter.txt
eye1.jpg eye2.jpg 0.525641
developer@ocularcloud-de:~/osiris/examples/matching/scores$
```

Fig. 14.3 Virtual machine implementing the developer interface. The web browser displays the developer documentation wiki. A terminal window shows the match score for two images as computed by a matching algorithm

14.4 An Operational Example

Ocular biometrics is the combination of multiple modalities in and near the eye region, such as the iris and periocular region [1, 25, 36]. In this operational example, we focus on the iris and periocular modalities as examples (together referred to as "ocular"). We performed two sets of experiments that illustrate the potential benefits of the framework. The first experiment shows that different algorithms provide different matching performance, thereby motivating the need for a system that supports a plurality of algorithms. The second experiment evaluates the performance of the system when the algorithm is automatically selected.

14.4.1 An Illustration of Algorithmic Diversity

The dataset used for the first experiment is the ND-IRIS-0405 iris dataset [6]. This dataset contains 64,980 images corresponding to 356 unique subjects and 712 unique irises. For our evaluation, we use iris images of 12 subjects and 12 images of the same iris per subject. In total, we used 144 images. All of the matching algorithms used by the example system are based on Open Source for IRIS (OSIRIS), which is a well-known open-source iris recognition system developed in the framework of the BioSecure project.[3] Specifically, OSIRIS *4.1* was used,[4] which is composed of four processing modules—segmentation, normalization, encoding, and matching. Gabor filters are applied to the normalized iris image, and the resulting phasor responses are quantized into a binary feature set. The Hamming distance measure is used to compare the binary feature sets of two iris images in order to obtain the final matching score.

In order to mimic the use of multiple algorithms, different Gabor filter parameters were selected for the OSIRIS algorithm, resulting in different sets of Gabor filters. This was accomplished by changing the *sizes* of the Gabor filters, or by changing the *number* of Gabor filters. Gabor filter coefficient sizes are defined in terms of the coefficient matrix ($m \times n$). We used five different Gabor filter parameter sets **A, B, C, D**, and **E** for this experiment. The **A, B**, and **C** parameter sets have 2 Gabor filters each, with coefficient matrix sizes of 9×15, 9×27, and 9×51, respectively. Parameter sets **D**, and **E** have 4 and 6 Gabor filters each, respectively. When used with OSIRIS, each parameter set is viewed as a different algorithm, which we denote Algorithm **A**, Algorithm **B**, Algorithm **C**, Algorithm **D**, and Algorithm **E**.

The following experiment was performed to evaluate and compare the performance of the three algorithms **A**, **B**, and **C**. False accept rate (FAR), false reject rate (FRR), and genuine accept rate (GAR) are computed for the test dataset of 144 images. Based on the number of subjects ($N = 12$) and the number of images ($t = 12$) per subject, we obtain $Nt(t-1)/2 = 792$ genuine scores and $(N(N-1)t^2)/2 = 9504$ imposter scores. The ROC (GAR vs FAR) curve at various threshold points for the first experiment is shown in Fig. 14.4. Algorithm **B** with 2 Gabor filters of size 9×27 performs marginally better than the other two algorithms. But it can be observed from the curves that there is no clear winner. However, these curves suggest that different algorithms may be needed depending upon operational requirements of FAR and/or GAR.

A similar experiment was performed for comparing the algorithms (**A, D**, and **E** which have a different number of filters (2, 4, and 6, respectively). The ROC curve for this experiment is shown in Fig. 14.5. This figure quite evidently solidifies the assumption and the motivation behind the solution proposed as we can clearly see that one algorithm never comes out on top as the curves do intersect at a number of points. So, again as already stated depending on the images, thresholds, and region of operation, a different algorithm can be selected for matching of the selfie ocular images.

[3]http://biosecure.wp.tem-tsp.eu/.

[4]http://svnext.it-sudparis.eu/svnview2-eph/ref_syst//Iris_Osiris_v4.1.

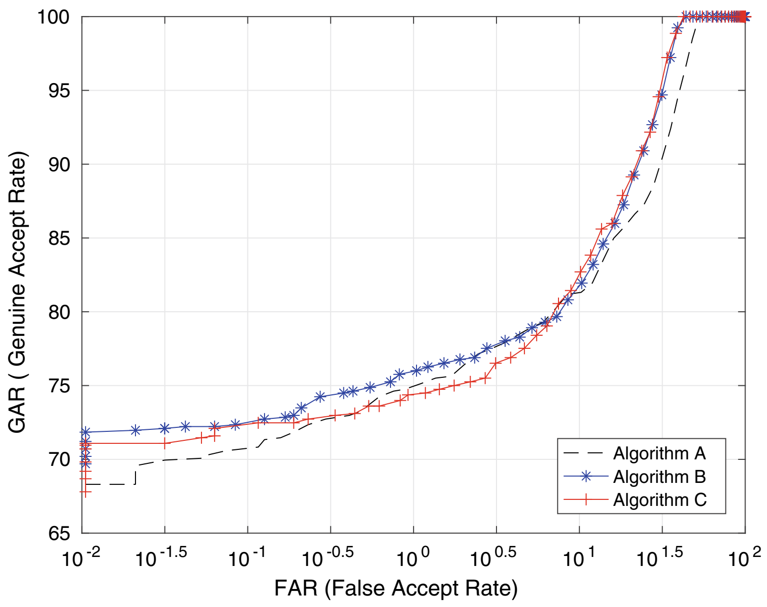


Fig. 14.4 ROC curves for three algorithms that each use two filters, but with different sizes

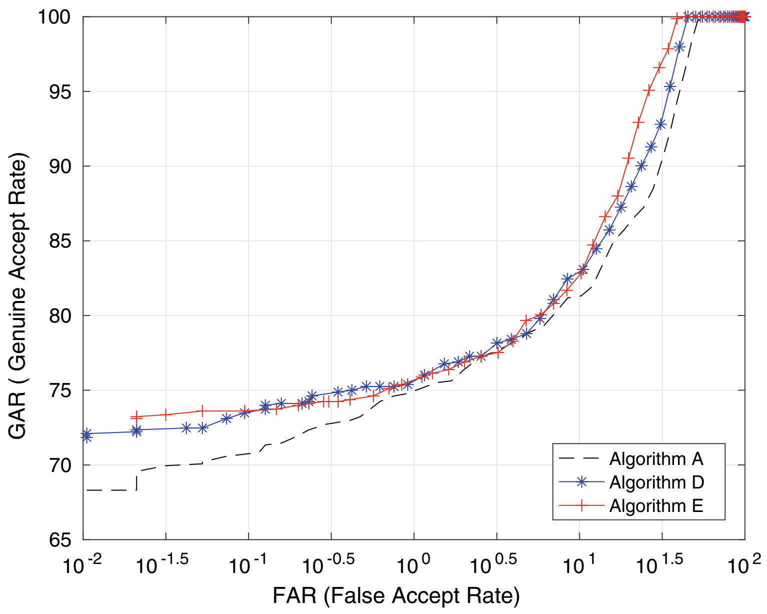


Fig. 14.5 ROC curves for three algorithms that each use a different number of Gabor filters

14.4.2 An Illustration of Automatic Algorithm Selection

In order to evaluate our SBaaS framework reference implementation, we conducted an experiment by using the web app on a smartphone. The trials were conducted by using a selfie image of the ocular region as one input and an ocular image from the phone gallery as the second input. Each trial entailed matching two images, and the experiment entailed 320 such trials. The trials consisted of both genuine and impostor image pairings.

Once the selfie images were uploaded to the cloud, the algorithm to be executed on the input images was selected automatically. Based on the input image characteristics, a particular algorithm was automatically invoked by the system at the time of authentication. In this experiment, three algorithms were used. The first was an OSIRIS-based iris recognition algorithm ("OSIRIS"), the second was a custom periocular matching algorithm ("Periocular"), and the third was a neural network iris matcher ("NN"). The selection method first computes the radius of the iris region in the input ocular image and uses this to select one of the three algorithms. In particular, the range between minimum and maximum radius of the iris is divided into three parts using two thresholds. If the radius of the iris is below the first threshold, the algorithm "Periocular" is selected. If the radius of the iris is between the first and the second threshold, then "OSIRIS" algorithm is selected for execution. Finally, if the radius of the iris is above the second threshold, then "NN" algorithm is executed.

Table 14.1 gives the number of times each algorithm was executed during the 320 trials conducted in the experiment. Besides hosting three completely different algorithms, it is possible for the cloud to host several instances of the same algorithm, where the different instances use different parameters. The experiment conveys the main theme of the proposed framework, i.e., depending on the input images, a different algorithm is selected each time and the developer for that selected algorithm is rendered a micropayment. This experiment shows that the proposed framework is feasible and creates an innovative and competitive ecosystem that benefits both software developers and end users.

For the above experiments, the radius of the iris has been considered as one of the variables to be used for automatic selection of the algorithm. However, there are a lot of other variables that can be used to automate the selection of the algorithm to be executed on the pair of input selfies. An example of other variable to be used for

Table 14.1 Table showing the number of executions for each algorithm for a total of 320 trials

Serial No.	Developer name	Algorithm name	Modality	Number of executions	Number of executions in %
1	TMPS	OSIRIS	Iris	102	31.9
2	ROSS	Periocular	Ocular	68	21.2
3	VT	NN	Iris	154	46.9

automation could be the matching score generated by the algorithm. An algorithm that gives the best matching score can be selected and the micropayment can be rendered to the corresponding developer. However, this entails running all the algorithms in the cloud on a given pair of inputs which would make the system really slow as it would lead to a huge computational cost.

Another example to automate the selection could be dependent on the micro-payments that are being rendered. Assuming that the micropayment amount for an algorithm is decided by the developer uploading their algorithm (it could be the licensing fee for using the algorithm). In that case, the algorithm selection could be automated based on the micropayment being rendered. Less micropayment algorithm could be selected more number of times.

Another way could be to select the strategy for automatic selection of the algorithm dynamically depending on the load of the system. If there is a heavy load and the number of selfie images to be matched in the queue is above a threshold, then the algorithm selection could be switched from the original automation strategy to a new strategy. The new strategy could be dependent on the computational cost of each algorithm, and this could be used to generate faster matching scores and reduce the load.

14.5 Secure Selfie-Biometrics-as-a-Service

In the general SBaaS framework discussed in Sec. 14.2, the enrollment selfie is stored in the biometric database. As in the reference implementation, the biometric database could simply be the image gallery on the user's device. Alternatively, the biometric database could be stored in the cloud or at a private server hosted by a bank, an e-commerce site, or a third-party authentication service provider. In these alternate scenarios, the security of the stored biometric data is critical and is not sufficiently integrated into the previously discussed SBaaS framework. In this section, we present an improved model that provides a solution for the security of the stored biometric data. This new model is termed "Secure Selfie-Biometrics-as-a-Service (SSBaaS)." A general SSBaaS architecture is given in Fig. 14.6. The SSBaaS architecture consists of two modules: a *feature extraction module (FEM)* and a *biometric security module (BSM)*. The FEM consists of feature extraction algorithms uploaded by the developers, and the BSM consists of the common back-end components required to keep the biometric secure in the SSBaaS architecture.

14.5.1 Feature Extraction Module

The SSBaaS model differs from the previously discussed SBaaS model in terms of how the developer contributes to algorithms. In the SBaaS model, the developer uploads an end-to-end biometric matching algorithm to the cloud. However, in the

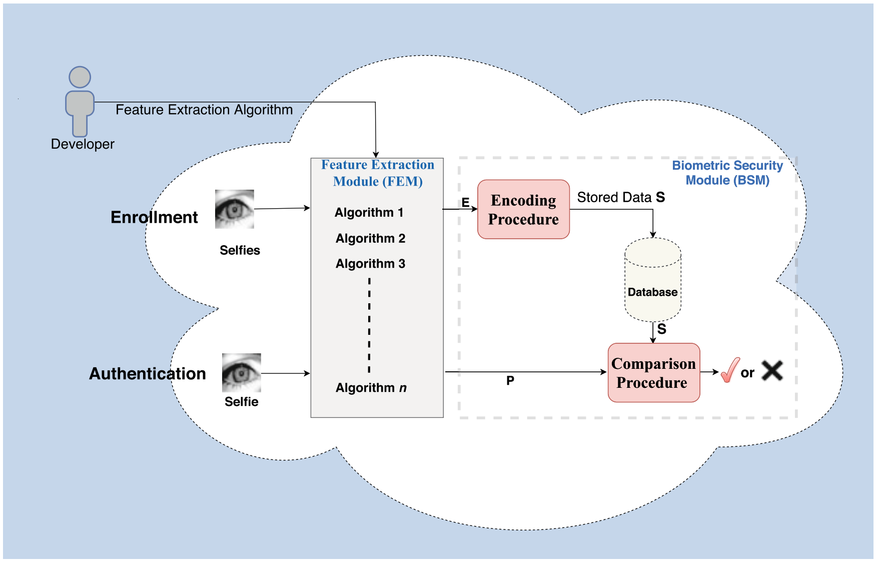


Fig. 14.6 General secure selfie-biometrics-as-a-service architecture

SSBaaS model, the developer only uploads algorithms for the extraction of features from the enrollment and probe selfies. The FEM contains all the feature extraction algorithms uploaded by the developers. The features extracted from each selfie have to be in the form of a binary vector. To preserve the privacy of the user, the binary feature vector is not directly stored in the database. Instead, the binary feature vector is passed through the BSM to generate a secure biometric template that is stored in the database. It is important to articulate to the developers the requirements that are enforced on the biometric feature extraction algorithms to generate the biometric feature vectors that are compatible for use in the BSM.

The statistical and privacy-preserving properties desired for the biometric feature vectors are as follows:

1. A bit in the feature vector is equally likely to be zero or one, which helps in maximizing the entropy of the feature vector.
2. A given bit in a feature vector provides no information about any other bit in the feature vector, which implies different bits in the feature vector are independent of each other.
3. The feature vector of one person provides no information about the feature vector of the other person, which implies inter-user independence.
4. Strong intra-user dependence, which implies different measurements of the same user are related by a binary symmetric channel (BSC) with crossover probability p where p is much smaller than 0.5.

Upon registering with the system, a developer is required to be bound to a set of constraints that enforce the above properties. Satisfying the above properties not only ensures good matching performance but also provides nice privacy-preserving features. This also benefits the developers by improving the chances of their algorithm being selected and micropayment being rendered to their accounts. However, designing feature vectors to possess such privacy-preserving properties forces a compromise between robustness and discriminability of the feature values, which in turn affects the accuracy (FRR and FAR) of the system, underscoring the fact that privacy at this time comes at the price of performance.

14.5.2 Biometric Security Module

In the SSBaaS model, it is assumed that the enrollment biometric information is stored in the cloud or on a private server. Consequently, the SSBaas model must preserve the biometric information confidentiality of the user. The leakage of biometric information stored in the cloud to an adversary constitutes a serious threat to security and privacy because if an adversary gains access to a biometric template, he can potentially obtain the stored user information. The attacker can use this information to gain unauthorized access to the system by reverse engineering the system and creating a physical spoof. Furthermore, an attacker can abuse the biometric information for unintended purposes and violate user privacy [21].

To alleviate such security and privacy concerns, secure biometric schemes have been developed to allow for authentication without requiring the enrollment biometric template to be stored in its raw format. BSM, which is shown as a part of Fig. 14.6, presents a general secure biometric scheme. The functionality of BSM is to develop a suitable *encoding procedure* for transforming enrollment biometric data into a template to be stored in the cloud and also to develop a *comparison procedure* for matching the probe biometric data with the stored template to produce an authentication decision. The BSM constitutes the back end of the SSBaaS architecture and is common to all developers and users of the complete system. All the feature extraction developers leverage the same BSM and have no direct access to the BSM. Rather, they only contribute to the FEM.

There are four main specific implementations of secure biometric schemes that are widely used: *fuzzy commitment, secure sketch, secure multiparty computation*, and *cancelable biometrics* [26]. *Fuzzy commitment* and *secure sketch* are biometric cryptosystem methods and are usually implemented with error-correcting codes and provide information-theoretic guarantees of security and privacy (e.g., [14, 15, 20, 23, 30]). *Secure multiparty computation* architectures are distance based and use cryptographic tools. *Cancelable biometrics*, which is a transformation based method, uses revocable and non-invertible user-specific transformations for distorting the enrollment biometric (e.g., [17, 27, 34, 37]), with the matching typically performed in the transformed domain. Fuzzy commitment, secure sketch, and cancelable biometrics architectures are described briefly below, treating each as a special manifestation of the BSM in 14.6.

Fuzzy commitment, a classical method of biometric protection, was first proposed by Juels and Wattenberg [15] in 1999. Fuzzy commitment is a key-binding method of biocryptosystem, and the encoding procedure involves combining a randomly generated vector $\mathbf{Z}$ with the enrollment biometric feature $\mathbf{E}$ resulting in the stored data $\mathbf{S}$. The comparison procedure checks whether the randomly generated vector $\mathbf{Z}$ is exactly recovered using the probe feature vector $\mathbf{P}$ and $\mathbf{S}$. There are many methods of implementing this fuzzy commitment scheme. However, a common method is to use error control coding (ECC). An example of using ECC for fuzzy commitment involves constructing the stored data as $\mathbf{S} = \mathbf{G}^T\mathbf{Z} \oplus \mathbf{E}$, where $\mathbf{G}$ is the generator matrix of an ECC. During authentication, the probe feature vector $\mathbf{P}$ is combined with $\mathbf{S}$ using $\mathbf{S} \oplus \mathbf{P}$. Next, using ECC decoding, the system attempts to decode the random message $\mathbf{Z}$ and allows access only if it is successful.

Secure sketch is a key generation method where some helper data or a sketch $\mathbf{S}$ is derived from the enrolled biometric feature vector $\mathbf{E}$ and stored in the database. The probe is given access when the probe biometric feature vector $\mathbf{P}$ is consistent with the stored secure sketch $\mathbf{S}$. The sketch $\mathbf{S}$ should be constructed so that it reveals little or no information about $\mathbf{E}$. Similar to fuzzy commitment, a common method of implementing secure sketch is to use ECC. In this method, ECC is applied to the biometrics or the feature vector to generate a sketch, which is stored in the database. The secure sketch $\mathbf{S}$ is constructed as $\mathbf{S} = \mathbf{HE}$; which is constructed as a syndrome of an ECC with parity check matrix $\mathbf{H}$. A legitimate probe biometric $\mathbf{P} = \mathbf{E}$ would be a slightly error-prone version of $\mathbf{E}$. Consequently, authentication can be accomplished by attempting to decode $\mathbf{E}$ given $\mathbf{P}$ and $\mathbf{S}$.

Cancelable biometrics involves transforming or distorting the enrollment biometric with a non-invertible user-specific transformation. The transformation in cancelable biometrics is a one-way transformation and can be applied either to the original biometric or in the feature domain. The advantage of using one-way transformations is that they are non-invertible and therefore the original biometric cannot be recovered easily. This transformation is revocable as well, which means that if the biometric is compromised, a new transformation can be applied to generate the cancelable template. This helps in protecting the privacy and also deters cross-matching since a different transformation can be used for a different application. Cancelable biometrics was first proposed by Ratha et al. [27], following which, there have been various different methods of generating cancelable biometric templates. Some of the popular methods use non-invertible transforms [27], bio-hashing [17], salting [37], and random projections [34]. Literature surveys on cancelable biometrics can be found in [26] and [24].

The secure biometric architectures explained above could be extended to include multiple biometric traits of a user [8, 21, 22, 31, 33]. Nagar et al. [21] developed a multimodal cryptosystem based on feature-level fusion using two different security architectures, fuzzy commitment, and fuzzy vault. In [31], face and fingerprint templates are concatenated to form a single binary string, and this concatenated string is used as input to a secure sketch scheme. In [33], a feature-level fusion framework is presented to generate a shared representation from each user's multiple biometrics. For each user, a selection of a different set of reliable and discriminative features from the shared representation is performed to generate a cancelable biometric template. This cancelable template is passed through an appropriate error-correcting decoder to find the closest codeword, which is hashed to generate the final secure multimodal template.

14.5.3 Reference Implementation of SSBaaS

A reference implementation of the SSBaaS architecture is shown in Fig. 14.7. In the reference implementation, the focus is on presenting a specific manifestation of the BSM. During enrollment, the user submits a selfie (i.e., enrollment selfie), which is transmitted to the cloud. In the cloud, depending on the learning or the state of the system, one or more feature extraction algorithms from the FEM are executed for the enrollment selfie. Initially, when the system has not learnt anything, all the feature extraction algorithms may be executed by the system for a given enrollment selfie. However, with an increase in enrollments, the system learns the best feature extraction algorithm for a given enrollment selfie, depending on certain learning criterion. Examples of these learning criteria are discussed later at the end of this section. For now, we can assume each feature extraction algorithm from the FEM is executed and one enrollment feature vector, denoted by E, is generated for each algorithm. Consequently, the number of enrollment feature vectors for each enrollment selfie is equal to the number of feature extraction algorithms (say n) in the FEM. For clarity of exposition, only one feature vector **E** is shown at the output of FEM in Fig. 14.7.

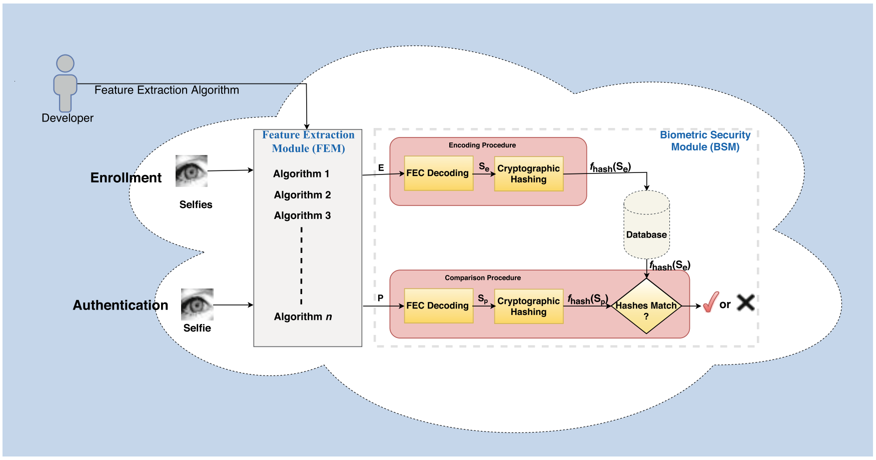


Fig. 14.7 Reference implementation of SSBaaS

The enrollment binary feature vector **E** is now passed to the BSM module for further processing and generation of secure biometric template in the database. The encoding procedure in this BSM consists of two steps: *forward error correction (FEC) decoding* and *cryptographic hashing*. The FEC decoding in this implementation is the equivalent of a secure sketch template protection scheme. In a secure sketch scheme, sketch or helper data are generated from the user's biometrics, and this sketch is stored in the access control database. There are many methods of implementing this secure sketch scheme. However, a common method is to use error control coding. In this method, error control coding is applied to the biometrics or the feature vector to generate a sketch which is stored in the database. Similarly, in this implementation, the FEC decoding is considered to be the error control coding part required to generate the secure sketch. The enrollment binary feature vector **E** generated from the FEM is considered to be the noisy codeword of some error-correcting code. This noisy codeword is decoded using FEC decoding, and the output of the decoding is the biometric secure sketch $\mathbf{S_e}$ that corresponds to the codeword closest to the enrollment feature vector. This biometric sketch $\mathbf{S_e}$ is cryptographically hashed to generate the secure biometric template $f_{\mathsf{hash}}(\mathbf{S_e})$, which is stored in the database. The same procedure of FEC decoding and cryptographic hashing is applied for all the n feature vectors generated by the n feature extraction algorithms for an enrollment selfie. This would imply that for each user's enrollment selfie, there would n cryptographic hashes stored in the database.

During authentication, the same process is performed. The user submits a probe selfie, which is transmitted to the cloud for authentication. A probe feature vector **P** is generated using the feature extraction algorithm in FEM. Next, the probe feature vector **P** is passed through an FEC decoder for the same error-correcting code used during the enrollment. The output of the FEC decoder is the probe biometric sketch $\mathbf{S_p}$, which is cryptographically hashed and access is granted only if this hash matches the enrolled hash. During authentication, if it is a genuine probe, the enrollment **E** and the probe vector **P** would usually decode to the same codeword in which case the hashes would match and access would be granted. Generally, if it is a legitimate probe, access would be granted. However, an adversary may use synthesized biometrics to fool and gain access to the system. Therefore, any analysis of the SSBaaS model must take into account not only authentication accuracy but also the information leakage and the possibility of attacking the system when the stored template is compromised.

Initially, when the system is still trying to determine the best feature extraction algorithm for a given user, the method discussed above could be one way of doing the enrollment, where the number of hashes stored per user is equal to the number of feature extraction algorithms in the FEM. Over a period of time, the system learns the best feature extraction algorithm for a given user depending on a number of variables. One of the variables is the execution time of the feature extraction algorithm. Some of the feature extraction algorithms may execute faster than the other algorithms. However, the execution time could be dependent on the resolution of the image, which in turn could be dependent on the device being used to capture the selfie. A *device table* could be stored in the cloud providing information as to which feature extraction algorithm works better with images from a particular device. For example, if the device is an iPhone 8, then Algorithm 3 may give fast and accurate results. In this case, iPhone 8 could be indexed with Algorithm 3. Using this table, the system can decide which algorithm needs to be used if the user operates a particular device; thus, it should be able to generate the enrollment feature vector only using the corresponding algorithms from the device table. However, this is just one method of deciding the best feature extraction algorithm. There could be other variables such as matching accuracy and micropayment cost that could be used to decide the best feature extraction algorithm for particular user enrollment. This is a design decision, and it might differ depending on the system requirements.

14.6 Summary

In this chapter, we presented a Selfie-Biometrics-as-a-Service framework for performing selfie biometric matching in a cloud environment using the sensors available in ordinary smartphones. The proposed biometrics-as-a-service paradigm enables users to perform biometric matching in a web interface. Moreover, the Platform-as-a-Service model enables the developers of recognition technology to upload their algorithms to the cloud. By selecting algorithms for execution and rendering micropayments to the corresponding developer, continuous algorithm innovation is encouraged. A reference implementation and an operational example have been presented demonstrating that the architecture is feasible in the form of a case study based on ocular recognition. Additionally, an overview of a secure Selfie-Biometrics-as-a-Service model has been discussed with a major focus on biometric template security in the cloud.

Acknowledgements This research was funded by the Center for Identification Technology Research (CITeR), a National Science Foundation (NSF) Industry/University Cooperative Research Center (I/UCRC).

References

1. Alonso-Fernandez F, Bigun J (2015) Near-infrared and visible-light periocular recognition with gabor features using frequency-adaptive automatic eye detection. IET Biom 4(2):74–89
2. Barra S et al (2015) Ubiquitous iris recognition by means of mobile devices. Pattern Recognit Lett 57:66–73
3. Beimborn D, Miletzki T, Wenzel S (2011) Platform as a service (PaaS). Bus & Inf Syst Eng 3(6)
4. Bharadi VA, D'silva GM (2015) Online signature recognition using software as a service (SAAS) model on public cloud. In: International conference on computer, communication and automated, pp 65–72
5. Bommagani AS, Valenti MC, Ross A (2014) A framework for secure cloud-empowered mobile biometrics. In: Proceeding of IEEE military communications conference, pp 255–261
6. Bowyer KW, Flynn PJ (2010) The ND-IRIS-0405 iris image dataset. University of Notre Dame, CVRL

7. Bradski G (2000) The opencv library. Dr. Dobb's J Softw Tools Prof Program 25(11):120–123 (2000)
8. Canuto AM, Pintro F, Xavier-Junior JC (2013) Investigating fusion approaches in multi-biometric cancellable recognition. Expert Syst Appl 40(6):1971–1980
9. Chow R, Jakobsson M, Masuoka R, Molina J, Niu Y, Shi E, Song Z (2010) Authentication in the clouds: a framework and its application to mobile users. In: Proceedings of the 2010 ACM workshop on cloud computing security workshop, CCSW '10. ACM, New York, NY, USA, pp 1–6. https://doi.org/10.1145/1866835.1866837
10. Das R (2013) Biometrics in the cloud. Keesing J Doc Identity, 21–23
11. de Freitas Pereira T, Marcel S (2015) Periocular biometrics in mobile environment. In: Proceeding of biometrics: theory, applications and systems (BTAS), pp 1–7
12. Jeong DS, et al (2006) Iris recognition in mobile phone based on adaptive gabor filter. In: Proceeding of international conference on biometrics (ICB), pp 457–463
13. Jillela RR, Ross A (2015) Segmenting iris images in the visible spectrum with applications in mobile biometrics. Pattern Recognit Lett 57(C):4–16
14. Juels A, Sudan M (2002) A fuzzy vault scheme. In: Proceeding IEEE international symposium on information theory, p 408. https://doi.org/10.1109/ISIT.2002.1023680
15. Juels A, Wattenberg M (1999) A fuzzy commitment scheme. In: Proceeding 6th ACM conference on computer and communications security, pp 28–36 (1999)
16. Kang JS (2010) Mobile iris recognition systems: an emerging biometric technology. Procedia Comput Sci 1(1):475–484
17. Kong A, Cheung KH, Zhang D, Kamel M, You J (2006) An analysis of biohashing and its variants. Pattern Recognit 39(7):1359–1368
18. Lawton G (2008) Developing software online with platform-as-a-service technology. Computer 41(6):13–15
19. Mell P, Grance T (2011) The NIST definition of cloud computing. In: Recommendations of the national institute of standards and technology, special publication pp 800–145
20. Nagar A, Nandakumar K, Jain AK (2008) Securing fingerprint template: fuzzy vault with minutiae descriptors. In: Proceeding 19th international conference on pattern recognition. https://doi.org/10.1109/ICPR.2008.4761459
21. Nagar A, Nandakumar K, Jain AK (2012) Multibiometric cryptosystems based on feature-level fusion. IEEE Trans Inf Forensics Secur 7(1):255–268. https://doi.org/10.1109/TIFS.2011.2166545
22. Nandakumar K, Jain AK (2008) Multibiometric template security using fuzzy vault. In: Proceeding IEEE international conference on biometrics: theory, applications and systems
23. Nandakumar K, Jain AK, Pankanti S (2007) Fingerprint-based fuzzy vault: implementation and performance. IEEE Trans Inf Forensics Secur 2(4):744–757. https://doi.org/10.1109/TIFS.2007.908165
24. Patel VM, Ratha NK, Chellappa R (2015) Cancelable biometrics: a review. IEEE Signal Process Mag 32(5):54–65. https://doi.org/10.1109/MSP.2015.2434151
25. Raghavendra R, Busch C (2016) Learning deeply coupled autoencoders for smartphone based robust periocular verification. In: 23rd international conference on image processing (ICIP). IEEE
26. Rane S, Wang Y, Draper SC, Ishwar P (2013) Secure biometrics: concepts, authentication architectures, and challenges. IEEE Signal Process Mag 30(5):51–64. https://doi.org/10.1109/MSP.2013.2261691
27. Ratha NK, Chikkerur S, Connell JH, Bolle RM (2007) Generating cancelable fingerprint templates. IEEE Trans Pattern Anal Mach Intell 29(4):561–572. https://doi.org/10.1109/TPAMI.2007.1004
28. Rose J (2016) Biometrics as a service: the next giant leap? Biom Technol Today 2016(3):7–9
29. Stojmenovic M (2012) Mobile cloud computing for biometric applications. In: 15th international conference on network-based information system, pp 654–659
30. Sutcu Y, Li Q, Memon N (2007) Protecting biometric templates with sketch: theory and practice. IEEE Trans Inf Forensics Secur 2(3):503–512

31. Sutcu Y, Li Q, Memon N (2007) Secure biometric templates from fingerprint-face features. In: Proceeding IEEE conference on computer vision and pattern recognition
32. Talreja V, Ferrett T, Valenti MC, Ross A (2018) Biometrics-as-a-service: a framework to promote innovative biometric recognition in the cloud. In: Proceeding IEEE international conference on consumer electronics (ICCE)
33. Talreja V, Valenti MC, Nasrabadi NM (2017) Multibiometric secure system based on deep learning. In: Proceeding IEEE global conference on signal and information processing, pp 298–302. https://doi.org/10.1109/GlobalSIP.2017.8308652
34. Teoh AB, Kuan YW, Lee S (2008) Cancellable biometrics and annotations on biohash. Pattern Recognit 41(6):2034–2044
35. Thönes J (2015) Microservices. IEEE Softw 32(1), 116, 113–115
36. Woodard DL, Pundlik S, Miller P, Jillela R, Ross A (2010) On the fusion of periocular and iris biometrics in non-ideal imagery. In: 20th international conference on pattern recognition (ICPR). IEEE, pp 201–204
37. Zuo J, Ratha NK, Connell JH (2008) Cancelable iris biometric. In: Proceeding IEEE international conference on pattern recognition, pp 1–4