Design of Portable Biometric Authenticators—Energy, Performance, and Security Tradeoffs

David D. Hwang, Student Member, IEEE, Ingrid Verbauwhede, Senior Member, IEEE

Abstract — *Biometrics have become a popular means for access control and authentication. As the processing power of embedded systems has grown, efforts have been made to perform biometrics locally on constrained devices such as smart cards. This paper presents the design and consumer application of a portable fingerprint biometric authenticator with the form factor of a key dongle (as an alternative to biometric smart cards). A thorough investigation has been performed to determine the tradeoffs between security, performance, and energy and to determine the secure partitioning between dongle and server. Such a device could be used for applications such as automotive access control, secure credit card payments, and related authentication scenarios*[1].

Index Terms — Authentication, Fingerprint Biometrics, Embedded Systems, Security.

I. Introduction

Authentication is becoming an increasingly important issue in modern society. In consumer applications as diverse as financial transactions, remote computer login, building access control, and keyless entry, it is extremely important to prove that a person is who he claims to be. In recent years, traditional methods for authentication such as passwords and PIN numbers have been shown to suffer flaws in security. Such flaws include forgotten or easily guessed passwords, PIN numbers written on the back of cards, etc. Therefore, alternative methods for authentication have been sought to alleviate these problems.

One alternative to traditional authentication is biometrics [1]. Biometrics is based on the fact that a person possesses certain characteristics—such as retinal patterns, fingerprint patterns, gait, etc.—that are biologically or behaviorally unique to an individual. It is this characteristic, rather than a forgettable code or password, which is used to corroborate a user's claimed identity. Because such characteristics are not easily forged, cannot be forgotten, and are not easily guessed, biometrics solves many of the problems that exist in traditional authentication. Fingerprint biometrics is by far the most popular and inexpensive form of biometrics used in consumer applications.

Another alternative has been the use of embedded systems (e.g. PDAs and cell phones) for user-server authentication. For example, the Infrared Data Association (IrDA) [2] has recently announced standards to facilitate wireless payment authentication using infrared transmission on cell phones. Instead of using a credit card with a signature to pay for an item, a user merely sends data over an infrared link from a cell phone to a card reader to make an electronic transaction. An embedded authentication system being widely deployed on toll roads is the radio frequency identification (RFID) tag [3]. Smart cards are also frequently used for financial transactions and consumer authentication purposes, as a replacement for magnetic stripe cards.

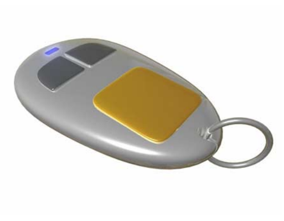


Fig. 1. Portable fingerprint biometric authenticator concept drawing.

In this paper we describe a device which combines both biometrics and embedded system authentication into a portable authenticator, as shown in concept in Fig. 1. The device contains a fingerprint sensor, a 32-bit RISC microcontroller, a Bluetooth wireless transceiver, and memory in the form factor of a key dongle. In the next section, we explain the consumer motivation of such a device and describe prior art.

II. Consumer Applications and Prior Art

A portable biometric authenticator has many advantages over a traditional user-to-server fingerprint verification system. In a traditional system, a user approaches a server (e.g. a building access station, a computer workstation, etc.) and enters a claimed identity either manually or via a magnetic card. After the user depresses a fingerprint on the server's sensor, the server performs a matching algorithm with a pre-stored template and allows access to the system based on a match or reject.

Though this type of verification system is the most common and simple, there are a number of potential security problems. One weakness is that fingerprints often leave a residue [4]; therefore a fingerprint can be lifted from the public sensor and a false finger can be generated from the lifted print. Another weakness is sociological, as users may be wary of storing their template at every bank, PC, employer, sports club, office, etc. that they interact with.

Thus we have designed the device concept in Fig. 1, which is a portable biometric authenticator. Such an authenticator

[1] This work was supported in part by the National Science Foundation (CCR-0098361) and the Fannie and John Hertz Foundation (DH).

The authors are with the Electrical Engineering Department, University of California—Los Angeles, Los Angeles, CA 90095 USA (email: dhwang@ee.ucla.edu, ingrid@ee.ucla.edu).

Contributed Paper
Manuscript received September 6, 2004

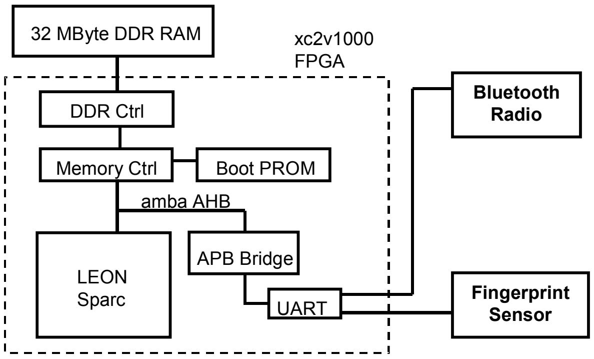


Fig. 2. Target system architecture and target FPGA test platform for the biometric authenticator. The architecture consists of a LEON Sparc V8 32-bit processor, memory, a Bluetooth radio, and a CMOS fingerprint sensor.

possesses its own fingerprint sensor and can also store the template and perform all biometrics locally, alleviating the problems of the server-based system. By using a challenge-response protocol and encrypting all data on the wireless channel, the communication between device and server remains secure. The device could be used in a variety of applications. For example, it could be used as an automotive access controller, which would not only open the doors, but also customize user-specific environment settings simultaneously. The authenticator could also be used in building access control for secure employee entry. Another potential application scenario is for financial and commercial transactions as a replacement for (biometric) smart cards, which is the most similar system to that described in this paper.

A biometric smart card is a processor-based device which can be used to authenticate a user with a server (e.g. the user's bank) via biometrics. There is much current research being performed on such devices [5]. There are several variants of such cards which can store the template only, perform feature extraction, or perform matching only.

There are two main differences between our authenticator and a biometric smart card. One potential drawback of biometric smart cards is the form factor, as inserting a fingerprint sensor on the device may not allow the card to conform to flexibility requirements of the smart card standard. We solve this drawback by using a key dongle form factor, popularized recently with key dongle flash memory. Furthermore, what differentiates this work with biometric smart cards is the link between the server (i.e. card reader) and device. A card reader to smart card link is often assumed to be a secure channel which is difficult to tamper with and eavesdrop. Our channel, however, is a wireless channel, which an attacker can easily eavesdrop or tamper with. Hence, additional cryptographic techniques are employed to provide confidentiality and integrity.

The primary design decisions to be made when creating such a device are biometric partitioning decisions. In other words, which biometric components should be located on the device versus on the server, and how do these decisions affect memory requirements, protocol latency, energy expenditure, and security. These metrics are extremely important because we are dealing with a portable embedded device with limited battery life, processing power, and memory capacity.

This paper will explain in depth such design decisions and their consequences in an embedded context. An examination of the security (not performance) aspects of different schemes using biometric smart cards is given in [6] and [7]. Design techniques for biometric smart cards are given in [8] and [9]. However, performance tradeoffs for metrics such as memory, latency, and energy—coupled with security analysis—of biometric verification is unique to this work. We address these factors in detail.

The remainder of the paper is organized as follows. A brief overview of fingerprint biometrics on an FPGA platform is presented in Section III. Section IV introduces six design alternatives for biometric partitioning. Section V presents simulation results and analysis of these alternatives for security, performance, energy, and memory. Section VI provides a design analysis summary, and Section VII provides concluding remarks.

III. Fingerprint Biometrics on an FPGA Platform

This paper deals with a biometric verification system, or a one-to-one match. In this system the user enters a claimed identity into a biometric processor and attempts to corroborate this identity with a candidate biometric. The biometric processor loads the pre-stored template associated with the claimed identity. After performing a match algorithm with the candidate biometric, the processor either accepts or rejects the user. A verification system is used in financial transactions and other consumer-based authentication systems and requires an a priori enrollment phase in which a template is first generated and pre-stored in the biometric processor. For the purposes of this paper, we assume the enrollment phase has already been completed. A verification system is in contrast to an identification system, or a one-to-many match, such as those used in criminal databases. There are four primary components in a verification system: the data collection subsystem, the feature extraction subsystem, the matching/decision subsystem, and the storage subsystem. Each of these components is discussed in detail in relation to our test platform, which is shown in Fig. 2.

The target platform used in simulation is an embedded system built around a soft-core LEON processor [10]. The processor is a 32-b RISC processor which is Sparc V8 compatible, and is synthesized to operate at 50 MHz on a Xilinx Virtex-II FPGA. The LEON core possesses an ALU, data and instruction caches, as well an AMBA bus structure containing interfaces with memory, two UARTs, and other peripherals interfaces. The first UART is connected to a Brainboxes Bluetooth wireless radio which operates over the RS-232 protocol, over which an socket IP protocol is overlaid. The second UART is connected to an Authentec AF-2 live-scan CMOS fingerprint sensor. The data and instruction memory are housed in 32-MByte DDR DRAM.

For the simulation results obtained in this paper, the protocol is implemented as a C program operating on LEON. To obtain accurate cycle counts for analysis, the protocols were simulated using the TSIM-LEON cycle-accurate instruction set simulator.

A. Data Collection

The data collection subsystem refers to the hardware required to obtain the fingerprint input from the user. In our system, we have used an Authentec image sensor to provide the raw image. The sensor uses capacitive differentials of a human fingerprint to obtain a raw bit map image of 256 x 256 pixels, with each pixel represented by one byte. Hence a raw image is a total of 65536 bytes. In this paper, the data collection is always performed on the device.

B. Feature Extraction

The purpose of feature extraction is to extract the unique characteristics of the fingerprint, which are called minutiae. The feature extraction is primarily a signal processing algorithm. Most feature extraction algorithms are designed in floating point format for operation on a workstation or PC. We have developed a fixed-point extraction algorithm which is optimized for operation on embedded devices. Starting with a baseline floating-point NIST standard reference algorithm, fixed-point refinement and optimization were accomplished for performance and memory.

The steps of the refined algorithm are illustrated in Fig. 3. The raw image is input into the extraction algorithm and a number of maps are generated (including ridge flow, quality, etc.). Based on these maps the image is binarized, that is, each pixel is mapped to either a black or white one-bit representation. A pattern search is performed on the binarized image to produce minutiae results, namely the type (ridge or bifurcation), direction (angle), and location to nearest neighbor (distance and angle). The output of the feature extraction algorithm is the set of all minutiae, usually between twenty to eighty for a typical fingerprint using our sensor. During the initial enrollment phase, this set is called the template minutiae set (or template, in short); during the verification phase, this set is called the candidate minutiae set. In this paper, the feature extraction algorithm can be performed either on the LEON or on a 500 MHz workstation.

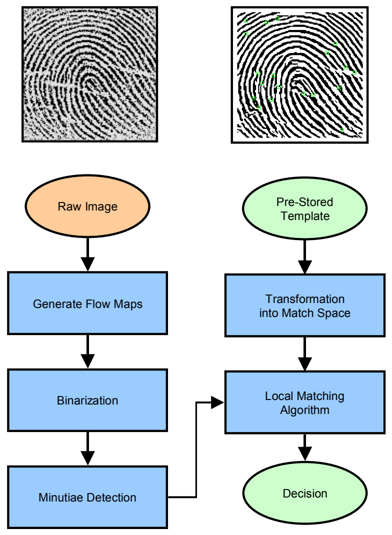


Fig. 3. Feature extraction and matching algorithms. This figure shows the steps required to extract candidate minutiae and compare them with a pre-stored template.

C. Matching / Decision

The matching algorithm is used during verification and compares the pre-stored template minutiae set with the candidate minutiae set. A new matching algorithm based on neighborhoods for embedded systems has been developed and is illustrated in Fig. 3. A reference point is first generated in the candidate minutiae set. Following this, the candidate set is aligned according to this point. Based on the correlation of the template with the candidate minutiae set, a matching score is produced. A pre-determined threshold decides if this score produces an acceptance or rejection of the user. In our final feature extraction plus matching system, we achieve a 0.5% false rejection rate (FRR) and a 0.01% false acceptance rate (FAR) [11]. Matching can be performed locally on the LEON or remotely on the workstation server.

D. Storage Subsystem

The storage subsystem is the physical means by which the template of the fingerprint is stored. In our platform, the template is stored either in the embedded device RAM or the server RAM. Our template maximally requires 2560 bytes for storage of one fingerprint.

IV. DESIGN ALTERNATIVES: BIOMETRIC PARTITIONING

As stated earlier, in the design of an embedded biometric authenticator, the primary top-level design decision that must be made is how to partition these four biometric components onto the device or server. As we shall demonstrate, performing the biometric processing on the server provides performance benefits with significant security problems. Performing all the biometric processing locally provides the best security, but

Scenario 1 (SC1)—Device data collection, server processing

Storage: Device and Server share key K.

Device→Server:	$ID_D \mid ID_S$
Server:	Verify $ID_D \mid ID_S$
	Generate RAND, SK = H_K(RAND | 0), SK' = H_K(RAND | 1), and $H_{SK'}$(RAND | ID_D | ID_S)
Server→Device:	RAND | $H_{SK'}$(RAND | ID_D | ID_S)
Device:	Generate SK = H_K(RAND | 0) and SK' = H_K(RAND | 1)
	Verify integrity via $H_{SK'}$(RAND | ID_D | ID_S)
	Obtain RAW IMAGE from user
Device→Server:	E_{SK}(RAW IMAGE) | $H_{SK'}$(RAW IMAGE)
Server:	Decrypt RAW IMAGE
	Verify integrity via $H_{SK'}$(RAW IMAGE)
	Perform FEATURE EXTRACTION algorithm
	Load TEMPLATE
	Perform MATCHING algorithm
	Perform DECISION algorithm to get RESULT
Server→Device:	E_{SK}(RESULT) | $H_{SK'}$(RESULT)
Device:	Decrypt RESULT
	Verify integrity via $H_{SK'}$(RESULT)

Scenario 2 (SC2)—Device data collection and storage, server processing

Storage: Device and Server share key K.

Device→Server:	$ID_D \mid ID_S$
Server:	Verify $ID_D \mid ID_S$
	Generate RAND, SK = H_K(RAND | 0), SK' = H_K(RAND | 1), and $H_{SK'}$(RAND | ID_D | ID_S)
Server→Device:	RAND | $H_{SK'}$(RAND | ID_D | ID_S)
Device:	Generate SK = H_K(RAND | 0) and SK' = H_K(RAND | 1)
	Verify integrity via $H_{SK'}$(RAND | ID_D | ID_S)
	Obtain RAW IMAGE from user
	Load TEMPLATE
Device→Server:	E_{SK}(RAW IMAGE | TEMPLATE) | $H_{SK'}$(RAW IMAGE | TEMPLATE)
Server:	Decrypt RAW IMAGE | TEMPLATE
	Verify integrity via $H_{SK'}$(RAW IMAGE | TEMPLATE)
	Perform FEATURE EXTRACTION algorithm
	Perform MATCHING algorithm
	Perform DECISION algorithm to get RESULT
Server→Device:	E_{SK}(RESULT) | $H_{SK'}$(RESULT)
Device:	Decrypt RESULT
	Verify integrity via $H_{SK'}$(RESULT)

Fig. 4. Protocols for scenario SC1 and scenario SC2. In these protocols, the server performs the feature extraction and matching/decision algorithms. They differ in the location of the template storage.

requires a relatively larger amount of energy and latency.

We have investigated the following six scenarios, or partitioning alternatives, as those we feel are most likely to be deployed in an actual authentication context. Each of the six partitioning alternatives is wrapped with a symmetric-key challenge/response protocol to ensure secure communications over the wireless channel and to allow for cryptographic authenticity verification. Table I provides a legend of the various cryptographic and biometric terminology used to describe the protocols. The remainder of this section will outline each of the partitioning strategies.

TABLE I

LEGEND OF FUNCTIONS AND DATA

Function or Data	Description
X | Y	Concatenation of data X and Y.
Y = E_{KEY}(X)	AES encryption of variable-length plaintext X with 128-b key KEY, producing ciphertext Y of the same length as X.
H_{KEY}(X)	Keyed-hash function (otherwise called a message authentication code algorithm) on variable-length data X using 128-b key KEY to produce a 128-b hash code. The code is generated using the cipher block chaining message authentication mode (CBC-MAC) of the AES cipher.
ID_D	128-b identifier of the device.
ID_S	128-b identifier of the server.
RAND	128-b random number.
K	128-b shared secret key.
SK = H_K(RAND | 0)	128-b encryption session key.
SK' = H_K(RAND | 1)	128-b hash function session key.
RAW IMAGE	65536 byte raw image collected from the sensor.
MINUTIAE	2560 byte candidate minutiae set.
TEMPLATE	2560 byte pre-stored fingerprint template.
RESULT	128-b server-generated value indicating a match, rejection, or other problem.

A. Case 1—Device Data Collection, Server Processing

The first scenario (SC1) is described in Fig. 4 and in Table II. This represents a scenario in which the device is only used to collect the raw image, and all biometric functions are performed on the server. Of all scenarios, this is the easiest to implement as no signal processing is performed on-device.

TABLE II

SCENARIO 1 (SC1)—DEVICE DATA COLLECTION, SERVER PROCESSING

Function	Device	Server
Data collection	X	
Feature Extraction		X
Storage		X
Matching/Decision		X

In this scenario, the device and the server share a 128-b secret key K. The secret key K can be an independent value or can be linked to the fingerprint template (e.g. K can be the result of applying a one-way hash function to the template.) The device first transmits its identifier (ID_D) together with the server's identifier (ID_S) to the server. The server verifies these identifiers, generates a random number RAND, and generates an encryption session key SK = H_K(RAND | 0), where the 0 denotes a 128-b zero vector. The server also generates a hash session key SK' = H_K(RAND | 1), where 1 denotes a 128-b vector of ones. The encryption and hash keys are different for stronger security. The server also generates a hash code of the random number and the two identifiers using the hash key SK'. It transmits RAND and this hash code to the device. After receiving this data, the device regenerates SK and SK', and verifies the hash code (thus verifying that the server possesses the correct key K). Upon verification, the device obtains the raw image from the user, encrypts this raw image with SK, creates a hash code of the raw image (to detect message tampering on the wireless channel), and transmits this to the server. The server decrypts the raw image and verifies the hash code. It performs all the biometrics and decides on a final result. It encrypts the final result and sends this to the device along with a hash code, simultaneously allowing or rejecting access to the system. The device decrypts the transaction result and verifies the hash code.

Scenario 3 (SC3)—Device processing, device storage

Storage: Device and Server share key K.	
Device$\rightarrow$Server:	$ID_D \mid ID_S$
Server:	Verify $ID_D \mid ID_S$
	Generate RAND, SK = H_K(RAND | 0), SK' =
	H_K(RAND | 1), and $H_{SK'}$(RAND | $ID_D \mid ID_S$)
Server$\rightarrow$Device:	RAND | $H_{SK'}$(RAND | $ID_D \mid ID_S$)
Device:	Generate SK = H_K(RAND | 0) and SK' = H_K(RAND | 1)
	Verify integrity via $H_{SK'}$(RAND | $ID_D \mid ID_S$)
	Obtain RAW IMAGE from user
	Perform FEATURE EXTRACTION algorithm
	Load TEMPLATE
	Perform MATCHING algorithm
	Perform DECISION algorithm
Device$\rightarrow$Server:	E_{SK}(DECISION) | $H_{SK'}$(DECISION)
Server:	Decrypt DECISION
	Verify integrity via $H_{SK'}$(DECISION)
	Generate RESULT
Server$\rightarrow$Device:	E_{SK}(RESULT) | $H_{SK'}$(RESULT)
Device:	Decrypt RESULT
	Verify integrity via $H_{SK'}$(RESULT)

Scenario 4 (SC4)—Device processing, server storage

Storage: Device and Server share key K.	
Device$\rightarrow$Server:	$ID_D \mid ID_S$
Server:	Verify $ID_D \mid ID_S$
	Generate RAND, SK = H_K(RAND | 0), and SK' =
	H_K(RAND | 1)
	Load TEMPLATE
	Generate E_{SK}(TEMPLATE) and $H_{SK'}$(RAND |
	TEMPLATE | $ID_D \mid ID_S$)
Server$\rightarrow$Device:	RAND | E_{SK}(TEMPLATE) | $H_{SK'}$(RAND | TEMPLATE
	| $ID_D \mid ID_S$)
Device:	Generate SK = H_K(RAND | 0) and SK' = H_K(RAND | 1)
	Decrypt TEMPLATE
	Verify integrity via $H_{SK'}$(RAND | TEMPLATE | $ID_D \mid ID_S$)
	Obtain RAW IMAGE from user
	Perform FEATURE EXTRACTION algorithm
	Perform MATCHING algorithm
	Perform DECISION algorithm
Device$\rightarrow$Server:	E_{SK}(DECISION) | $H_{SK'}$(DECISION)
Server:	Decrypt DECISION
	Verify integrity via $H_{SK'}$(DECISION)
	Generate RESULT
Server$\rightarrow$Device:	E_{SK}(RESULT) | $H_{SK'}$(RESULT)
Device:	Decrypt RESULT
	Verify integrity via $H_{SK'}$(RESULT)

Fig. 5. Protocols for scenario SC3 and scenario SC4. In these protocols, the device performs the feature extraction and matching/decision algorithms. They differ in the location of the template storage.

TABLE III

SCENARIO 2 (SC2)—DEVICE DATA COLLECTION AND STORAGE, SERVER PROCESSING

Function	Device	Server
Data collection	X	
Feature Extraction		X
Storage	X	
Matching/Decision		X

B. Case 2—Device Data Collection and Storage, Server Processing

Scenario SC2 is described in Fig. 4 and in Table III. SC2 represents a scenario similar to SC1 in that the device only serves to acquire the data; the server performs all biometric processing. However, in SC2 the device stores the template. This situation would be used in practice when a device is computationally-limited, yet for the aforementioned security reasons the template is stored on-device.

In this scenario, the device and server share secret key K. The initial steps of the protocol are the same as that for SC1. However, after the raw image is acquired, the device not only sends the image across the channel but also sends the template, both encrypted with SK. Similar to SC1, the data RAW IMAGE | TEMPLATE is hashed to produce a hash code, which insures data integrity on the channel. The server receives this data and decrypts the raw image and template. It then verifies the hash code of the data to insure the data was sent undisturbed. After this, the server performs the biometric feature extraction, matching, and decides on a final result. The transaction result is sent encrypted to the device along with a hash code.

C. Case 3—Device Processing, Device Storage

Scenario SC3 is described in Fig. 5 and in Table IV. This represents a scenario in which all biometric components—data collection, feature extraction, storage, and matching/decision—are performed locally on the device. If a device can be made tamper-proof in hardware and software, this presents a very secure solution [6], yet comes with a performance penalty (to be discussed).

TABLE IV

SCENARIO 3 (SC3)—DEVICE PROCESSING, DEVICE STORAGE

Function	Device	Server
Data collection	X	
Feature Extraction	X	
Storage	X	
Matching/Decision	X	

In this scenario, the device and server share secret key K. After the device sends the identifiers and the server sends its response, the device performs the biometrics. It first obtains the raw image from the user and performs feature extraction to obtain a candidate minutiae set. It then loads the template, which is stored on the device, and performs the matching and decision steps. Finally a decision is made. This decision is forwarded to the server along with a hash code. The server decrypts the decision and verifies the code. If satisfied, the server allows access to the system and sends the final transaction result to the device along with a hash of the result.

D. Case 4—Device Processing, Server Storage

Scenario SC4 is described in Fig. 5 and in Table V. SC4 represents a scenario in which all biometric processing is done on the device, just as SC3. However, the storage of the template is on the server rather than on the device. (In this sense, SC3 and SC4 are analogues of SC1 and SC2.) This situation would be used in cases where the device's memory is

Scenario 5 (SC5)—Mixed processing (device feature extraction)

Storage: Device and Server share key K.

Device$\rightarrow$Server:	$ID_D \mid ID_S$
Server:	Verify $ID_D \mid ID_S$
	Generate RAND, SK = H_K(RAND $\mid$ 0), SK' = H_K(RAND $\mid$ 1), and $H_{SK'}$(RAND $\mid$ $ID_D \mid ID_S$)
Server$\rightarrow$Device:	RAND $\mid$ $H_{SK'}$(RAND $\mid$ $ID_D \mid ID_S$)
Device:	Generate SK = H_K(RAND $\mid$ 0) and SK' = H_K(RAND $\mid$ 1)
	Verify integrity via $H_{SK'}$(RAND $\mid$ $ID_D \mid ID_S$)
	Obtain RAW IMAGE from user
	Perform FEATURE EXTRACTION algorithm
Device$\rightarrow$Server:	E_{SK}(MINUTIAE) $\mid$ $H_{SK'}$(MINUTIAE)
Server:	Decrypt MINUTIAE
	Verify integrity via $H_{SK'}$(MINUTIAE)
	Load TEMPLATE
	Perform MATCHING algorithm
	Perform DECISION algorithm to get RESULT
Server$\rightarrow$Device:	E_{SK}(RESULT) $\mid$ $H_{SK'}$(RESULT)
Device:	Decrypt RESULT
	Verify integrity via $H_{SK'}$(RESULT)

Scenario 6 (SC6)—Mixed processing (server feature extraction)

Storage: Device and Server share key K.

Device$\rightarrow$Server:	$ID_D \mid ID_S$
Server:	Verify $ID_D \mid ID_S$
	Generate RAND, SK = H_K(RAND $\mid$ 0), SK' = H_K(RAND $\mid$ 1), and $H_{SK'}$(RAND $\mid$ $ID_D \mid ID_S$)
Server$\rightarrow$Device:	RAND $\mid$ $H_{SK'}$(RAND $\mid$ $ID_D \mid ID_S$)
Device:	Generate SK = H_K(RAND $\mid$ 0) and SK' = H_K(RAND $\mid$ 1)
	Verify integrity via $H_{SK'}$(RAND $\mid$ $ID_D \mid ID_S$)
	Obtain RAW IMAGE from user
Device$\rightarrow$Server:	E_{SK}(RAW IMAGE) $\mid$ $H_{SK'}$(RAW IMAGE)
Server:	Decrypt RAW IMAGE
	Verify integrity via $H_{SK'}$(RAW IMAGE)
	Perform FEATURE EXTRACTION algorithm
Server$\rightarrow$Device:	E_{SK}(MINUTIAE) $\mid$ $H_{SK'}$(MINUTIAE)
Device:	Decrypt MINUTIAE
	Verify integrity via $H_{SK'}$(MINUTIAE)
	Perform MATCHING algorithm
	Perform DECISION algorithm
Device$\rightarrow$Server:	E_{SK}(DECISION) $\mid$ $H_{SK'}$(DECISION))
Server:	Decrypt DECISION
	Verify integrity via $H_{SK'}$(DECISION)
	Generate RESULT
Server$\rightarrow$Device:	E_{SK}(RESULT) $\mid$ $H_{SK'}$(RESULT)
Device:	Decrypt RESULT
	Verify integrity via $H_{SK'}$(RESULT)

Fig. 6. Protocols for scenario SC5 and scenario SC6. In these protocols, the biometric processing is split into two components. For SC5, the feature extraction is on the device and the matching is on the server. For SC6, the feature extraction is on the server and the matching is on the device.

untrusted, while the server database is trusted.

TABLE V

SCENARIO 4 (SC4)—DEVICE PROCESSING, SERVER STORAGE

Function	Device	Server
Data collection	X	
Feature Extraction	X	
Storage		X
Matching/Decision	X	

In this scenario the device and server share secret key K. The protocol proceeds as follows. After the device transmits the identifiers, the server produces the usual RAND and hash code. In addition, the server loads the template and encrypts this template with the key SK. It then transmits RAND, the encrypted template, and a keyed-hash of (RAND | TEMPLATE | ID_D | ID_S) to the device. The device generates the session keys and decrypts the template. Next, it verifies the integrity of the transmitted data via the hash code. After this, it obtains the raw image from the user and performs the feature extraction and matching/decision algorithms. It sends the encrypted decision and its hash to the server. The server verifies this decision and sends the final transaction result to the device.

E. Case 5—Mixed Processing (Device Feature Extraction)

Scenario SC5 is described in Fig. 6 and in Table VI. In this scenario, we attempt to split up the biometric processing elements for performance evaluation by assigning the feature extraction to the device and the matching/decision to the server. This scenario would be deployed in cases where the server database is trusted, and the server is required to make the final decision.

TABLE VI

SCENARIO 5 (SC5)—MIXED PROCESSING (DEVICE FEATURE EXTRACTION)

Function	Device	Server
Data collection	X	
Feature Extraction	X	
Storage		X
Matching/Decision		X

The device and server share secret key K. The device begins by sending the identifiers to the server, and the server responds by sending the usual RAND and hash code. After verifying the hash code, the device obtains the raw image from the user. It then directly performs the feature extraction algorithm on this raw image and sends the candidate minutiae set to the user in encrypted format, along with a hash of this set. Note that this saves on transmission cost, as sending a raw image requires transmitting 65536 bytes whereas sending the minutiae set requires a transmission of only 2560 bytes. The server decrypts the minutiae and verifies the hash code (to detect tampering of data). It then loads the template, performs the matching and decision and sends the final transaction result encrypted to the device.

F. Case 6—Mixed Processing (Server Feature Extraction)

The final scenario (SC6) is described in Fig. 6 and in Table VII. This scenario is the analogue of SC5. In this scenario, we again split the biometric processing elements; however, the feature extraction is performed on the server whereas the matching/decision is performed on the device.

TABLE VII
SCENARIO 6 (SC6)—MIXED PROCESSING (SERVER FEATURE EXTRACTION)

Function	Device	Server
Data collection	X	
Feature Extraction		X
Storage	X	
Matching/Decision	X	

In this scenario, the device and server share secret key K. The device begins the transaction by sending the identifiers to the server. The server responds by sending the RAND and the usual hash code. Upon verifying the hash code, the device obtains the raw image from the user and encrypts this (and hashes it) and sends this to the server. The server decrypts the raw image and verifies the hash code for tampering, and then proceeds to perform the feature extraction algorithm to obtain the candidate minutiae set. It then sends this encrypted set with its hash to the device. The device decrypts the minutiae set and verifies the integrity via the hash code. It then loads the stored template set and performs the matching/decision algorithm. It sends the decision and its hash to the server. The server verifies the decision and hash, and finally accepts or rejects the user and sends the transaction result to the device.

V. COST ANALYSIS: SECURITY AND PERFORMANCE TRADEOFFS

The device can be implemented as one of the aforementioned design alternatives. In this section, we perform a comparative analysis to see how the alternatives affect the device in terms of security, device cycle count, transmission bytes, energy expended, latency of the protocol, and device memory requirements.

A. Security Analysis

To analyze the security of each of the alternatives, we describe five attacks on the system and describe which of the alternatives are vulnerable to each attack. The first attack we consider is a *replay attack*. In a replay attack, a passive attacker would record a transmission on the channel and later rebroadcast this, masquerading as an authentic party. For example, an attacker can hear the message from the server to the device of

$$\text{RAND} \mid H_{SK'}(\text{RAND} \mid ID_T \mid ID_S)$$

and store this value. Later, the attacker can masquerade as the server and use this value to respond to the device's initial query. Since the device has no means to check if this RAND has been used before and if it is sent by the genuine server, the device will consider the attacker as an authentic server. All six scenarios are susceptible to this attack. This attack can be remedied by using a sequence number SQN which automatically increments both on the server and the device at each message. Hence, the server would now send:

$$\text{RAND} \mid H_{SK'}(\text{RAND} \mid \text{SQN} \mid ID_T \mid ID_S).$$

Based on the sequence number, the authenticity of the server can be determined by the device. A resynchronization procedure would be required if the device and server legitimately lose synchronization. A second attack on the system is a *server template storage attack*. In this attack, the active attacker attempts to hack the database of the server (i.e. a bank) to obtain the template of the user by out-of-bounds methods. This attack is similar to an internet attack on a bank's database of credit card numbers. Template attacks (both server and device) are important due to the fact that biometric characteristics cannot be replaced and that they are of limited number. Server template storage attacks are only valid against those scenarios in which the template is stored at the server, namely SC1, SC4, and SC5. With the template stored at the server, there are the additional sociological ramifications described in Section II that also must be taken into account in these scenarios. These attacks must be remedied by physical and software protection of the large consumer fingerprint databases. Server template attacks may be expensive to mount but also reward the attacker with not only a single fingerprint, but potentially all fingerprints in the database.

A third attack on the system is a *malicious server* attack. In this attack, the server does not store the template. However, if matching is performed on the server, the device must transmit the template to the server for matching; a malicious server would save this template and use it later for unscrupulous purposes. Similarly, if feature extraction is performed on the server, the malicious server would store the extracted candidate minutiae set. The malicious server attack is similar to the server template storage attack; in both cases, sensitive data is eventually stored on the server. The malicious server attack can be carried out on scenarios SC2 and SC6. The only scenario that is safe from both the server template storage attack and the malicious server attack is SC3, in which all biometrics are done on device.

A fourth attack on the system is the *device template storage attack*. In this attack, an attacker either steals a device or obtains a device lost by the user. He then proceeds to hack the device physically to obtain the memory component that houses the key and the template, thus compromising it. The scenarios that are vulnerable to this attack are SC2, SC3, and SC6. The device template storage attack can be thwarted by using tamper-resistance techniques [12] (such as zeroing the memory when an attack is sensed) and other techniques to make the memory secure.

A fifth attack on the system is a *device bypass attack*. In this attack, the device itself is hacked at the software level to always produce a valid authentication. In other words, the biometric functions of the device (fingerprint acquisition, feature extraction, etc.) are bypassed. The device always falsely produces a value that indicates a match has been made. This attack essentially uncouples the user from the device and allows anyone to use the device to authenticate with the server. The scenarios prone to this attack are the ones in which the decision is on the device side, namely SC3, SC4, and SC6.

The device bypass attack can be addressed by the design of secure instruction sets at the micro-architecture level. Table VIII summarizes the scenarios susceptible to the various attacks.

TABLE VIII
SUMMARY OF SECURITY ATTACKS

Attack	Affected Scenarios
Replay	SC1, SC2, SC3, SC4, SC5, SC6
Server Template Storage	SC1, SC4, SC5
Malicious Server	SC2, SC6
Device Template Storage	SC2, SC3, SC6
Device Bypass	SC3, SC4, SC6

B. Active Computation Cycle-Count Analysis

In this section we compare the protocols based on active computation cycles. An active computation cycle is defined as a cycle in which the LEON processor is actively using its resources to perform computation. This is in contrast to a passive computation cycle in the LEON, where the processor is waiting for a response for the server or is otherwise idle.

Table IX shows the total active computation cycles for each of the scenarios. The first number shows the number of cycles and the second number shows the relative percentage of that column to the overall active cycle count. Protocol cycles refer to the cycles required for the protocol, such as socket setup and comparison operations. Cryptography cycles include those for encryption and hashing. Feature extraction cycles include the number of cycles required for the feature extraction algorithm, and matching/decision cycles refer to the cycles required to perform matching and generate a final acceptance or rejection. The results can also be seen in Fig. 7.

TABLE IX
LEON ACTIVE COMPUTATION CYCLES AND RELATIVE COMPOSITION

Scenario	Protocol	Cryptography	Feature Extraction	Matching / Decision	TOTAL ACTIVE
SC1	1.67×10^6 (6.71 %)	2.32×10^7 (93.29 %)	-	-	2.49×10^7
SC2	1.68×10^5 (6.33 %)	2.49×10^7 (93.67 %)	-	-	2.66×10^7
SC3	1.63×10^5 (0.03 %)	6.59×10^5 (0.14 %)	4.58×10^8 (96.37 %)	1.64×10^7 (3.45 %)	4.75×10^8
SC4	3.99×10^5 (0.08 %)	9.36×10^6 (1.90 %)	4.69×10^8 (95.37 %)	1.30×10^7 (2.65 %)	4.92×10^8
SC5	2.44×10^5 (0.05 %)	1.74×10^6 (0.37 %)	4.69×10^8 (99.58 %)	-	4.71×10^8
SC6	2.06×10^6 (4.37 %)	2.53×10^7 (53.64 %)	-	1.98×10^7 (41.99 %)	4.71×10^7

From the table, it is clear that the most dominant consumer of cycles is the feature extraction algorithm. On the scenarios that perform local feature extraction (SC3, SC4, and SC5), between 96% and 99% of the cycles are used to perform this function, with a final active computation count on the order of 10^8. For the scenarios that do not require local feature extraction (SC1, SC2, and SC6), the number of active cycles is an order of magnitude lower, on the order of 10^7.

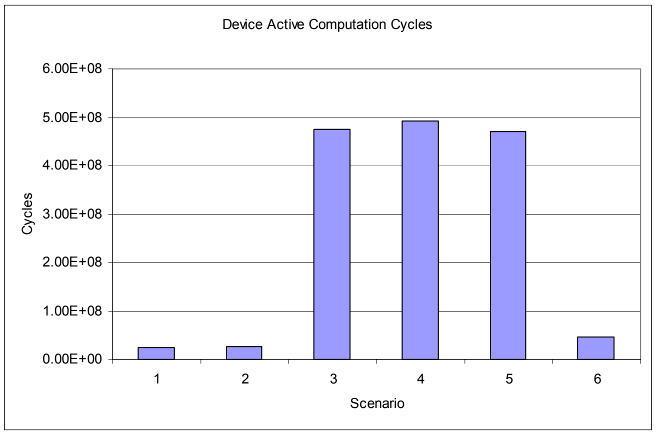


Fig. 7. Device active computation cycles. The number of cycles is dominated in SC3, SC4, and SC5 by the feature extraction algorithm.

TABLE X
TRANSMISSION OF DATA BETWEEN SERVER AND DEVICE

Scenario	Bytes Transmitted	Bytes Received	TOTAL
SC1	65584	64	65648
SC2	68144	64	68208
SC3	64	64	128
SC4	64	2624	2688
SC5	2608	64	2672
SC6	65584	2640	68224

C. Data Transmission Analysis

The number of bytes transmitted in each protocol is shown in Table X. Scenarios SC1, SC2, and SC6 require the largest number of bytes to be transmitted, as the encrypted fingerprint is sent over the wireless channel. The greatest number of bytes received is in SC4 and SC6, which require the template to be sent from the server to the device. The number of bytes transmitted affects both the energy of the device and the latency of the protocol, as discussed in the ensuing sections.

D. Energy Analysis

In this section we analyze the energy requirements of the device for each protocol. Energy is an extremely important metric because we assume our embedded device will be battery operated and hence is able to perform a limited number of authentications. The total device energy is composed of the device active computation energy (ACE), the device passive computation energy (PCE), and the device communication energy (CE). These are calculated as follows.

We estimate the LEON FPGA soft-core energy by the estimate of 10mW/MHz. This coarse-grain estimate is based on the average behavior of LEON using Xilinx xpower, a toggle counter. Therefore our processor at 50 MHz consumes 500 mW of energy. To be conservative, this number is used for both active and passive computation cycles, as LEON was not in sleep mode while waiting for server transmissions. Therefore, the energy per cycle is 500 mJ/s $\times$ 50^{-1} s/Mcycle = 1×10^{-8} J/cycle. We calculate ACE and PCE in Joules as:

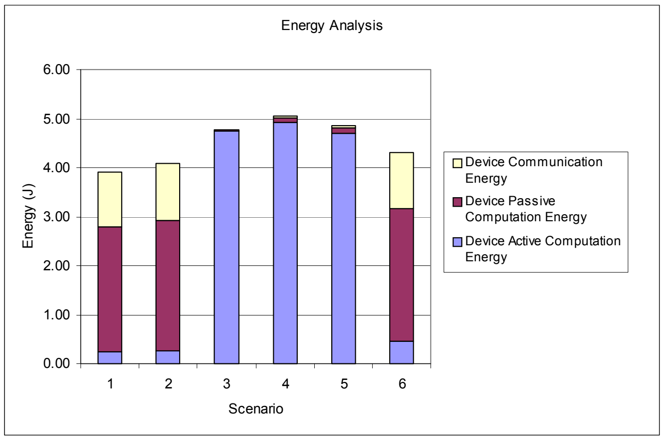


Fig. 8. Device energy analysis. The figure shows the energy consumed by the device for different partitioning scenarios.

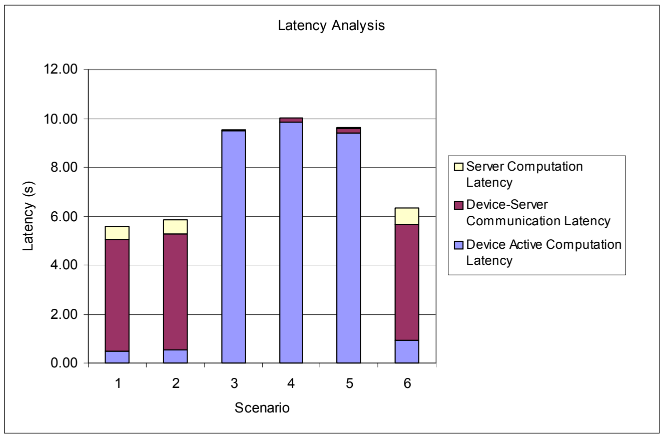


Fig. 9. Device latency analysis. The graph depicts the total number of cycles required to complete the entire authentication protocol.

$$\text{ACE} = 1 \times 10^{-8} \text{ J/cycle} \times \text{active cycles}$$
$$\text{PCE} = 1 \times 10^{-8} \text{ J/cycle} \times \text{passive cycles.}$$

To calculate CE, we use the following manufacturer energy estimates. Assuming a 2 meter separation with the Bluetooth device transmitting and receiving at 115,000 b/s, the current drawn from the supply is 48.7 mA and 44.1 mA for transmission and reception, respectively. We calculate the energy per bit required to transmit as 48.7 mA $\times$ 5 V $\times$ $115{,}000^{-1}$ s/b = 2.11×10^{-6} J/b. For reception, the energy per bit is 44.1 mA $\times$ 5 V $\times$ $115{,}000^{-1}$ s/b = 1.91×10^{-6} J/b. Hence the device communication energy (CE) in Joules as:

$$\text{CE} = 2.11 \times 10^{-6} \text{ J/b} \times \text{bytes transmitted} \times 8 \text{ b/byte}$$
$$+ 1.91 \times 10^{-6} \text{ J/b} \times \text{bytes received} \times 8 \text{ b/byte.}$$

The total device energy is thus ACE + PCE + CE. The energy results for each of the scenarios can be seen in Fig. 8. As the results show, the largest consumer of cycles is the active consumption cycles due to the feature extraction algorithm, as described in the active cycle count analysis. For those scenarios that do not perform feature extraction, the largest consumer of cycles is the cycles expended as the transmission of the raw image takes place over the channel, expending radio energy and passive computation energy.

E. Latency Analysis

Since this device will be used in consumer scenarios in real-time, latency is another important design factor. Latency is defined as the total number of seconds expended from the beginning of the protocol to its completion. Latency is composed of three components: device active computation latency (ACL), device-to-server communication latency (DSCL), and server computation latency (SCL). (Note that the device-to-server communication latency plus the server computation latency make up the factor we call passive computation cycles.) The device active computation latency is defined as the number of seconds per cycle multiplied by the number of active computation cycles, hence:

$$\text{ACL} = (1/50 \times 10^{8}) \text{ s/cycle} \times \text{active cycles.}$$

The device-to-server computation latency is calculated as the time required to transmit a single bit multiplied by the number of bytes transmitted and received by the device, hence:

$$\text{DSCL} = 115{,}000^{-1} \text{ s/b} \times 8 \text{ b/byte} \times$$
$$(\text{bytes transmitted} + \text{bytes received}).$$

The server computation latency (SCL) is the number of seconds required for the server to perform its local computations, which differ according to different scenarios. The UNIX gprof tool was used for these simulations to obtain estimates of computation time on a 500 MHz workstation. The total latency of the protocol is thus ACL + DSCL + SCL, which is shown on Fig. 9. The latency of the protocol for our limited 50 MHz device ranges from 5.58 seconds to 10.03 seconds. Scenarios SC3, SC4, and SC5 require the most latency due to the processing time required for the feature extraction algorithm.

F. Memory Analysis

Memory analysis is performed using the Atomium memory profiling tool [13]. This tool requires that a particular C program be instrumented into an Atomium-compatible format and re-compiled using Atomium include files. The results of the program give peak memory information and access for the RAM.

The results of our Atomium analysis are shown in Fig. 10. The figure demonstrates that the largest memory is required for the feature extraction algorithm of SC3, SC4, and SC5. The memory is primarily consumed during the map generation operations, as the algorithm obtains spatial information about the entire fingerprint before pattern matching. The maximum memory required is 1.04 MB for scenario SC4 and the least RAM required is 208 KB for scenario SC1.

VI. DESIGN ANALYSIS SUMMARY

Final design decisions for embedded biometric devices therefore involve significant tradeoffs between performance and security metrics. In terms of security, we conclude that storing the template at the server (such as in SC1, SC4, and SC5) is not ideal due to server storage attacks as well as the sociological issues involved. The remaining scenarios are SC2, SC3, and SC6. However, both SC2 and SC6 are prone to the malicious server attack (resulting in the same problems as the

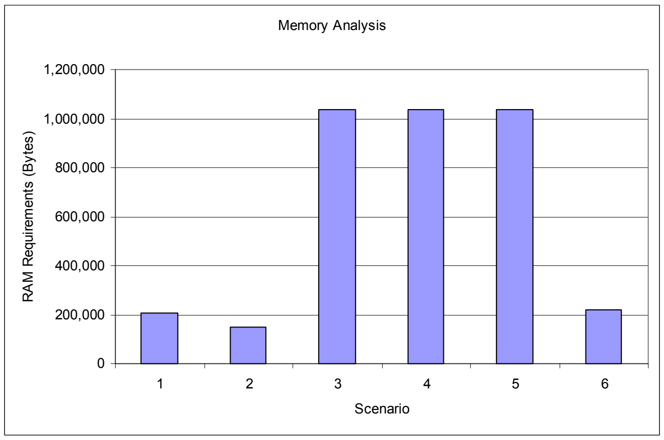


Fig. 10. Memory analysis. This graph shows the peak RAM memory requirements for each scenario.

server storage attack). Hence, for maximum security from this perspective, all biometric components should be performed on-device, as in SC3. Care must be taken to insure the device is tamper-proof in both hardware and software in this scenario.

In terms of performance, the greatest consumer of energy, cycle time, and memory in the device is the feature extraction algorithm, which consumes 96% to 99% of the cycles of the respective protocols, on the order of 10^8 cycles. (In contrast, the matching algorithm requires an order of magnitude fewer operations, on the order of 10^7 cycles.) Thus, the scenarios which require device feature extraction (SC3, SC4, and SC5) require the most energy and memory; those that use the server for feature extraction (SC1, SC2, and SC6) are more efficient in regard to these metrics. In general, the most efficient scenario is SC1, where the device is used only to transmit the encrypted raw image.

Different applications may place varying emphasis on security and performance. In our particular application, focusing on secure transactions, security is the most important metric and our final implementation is SC3, with all biometrics performed on-device.

VII. Conclusion

In this paper we have examined the design choices required to implement a portable biometric authenticator. Such choices focus around top-level decisions on how to partition the biometric functions between a device and server. Our results have shown that in terms of performance, off-loading the data to the server is the most efficient, yet produces a number of security problems. In terms of security, implementing all biometrics on-device is the most secure solution, but one with a performance downside. However, for high-security consumer applications (such as medical or financial applications) this is the solution that should be implemented.

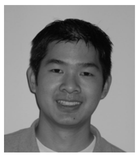
Future work in this area includes optimizing embedded feature extraction algorithms, developing techniques to allow for secure non-volatile storage for the template or other secure data.

Acknowledgment

The authors wish to thank Patrick Schaumont, Yi Fan, Shenglin Yang, Alireza Hodjat, Bocheng Lai, and Kazuo Sakiyama for their contributions and Danny De Cock of K.U. Leuven for feedback on security issues.

References

[1] S. Prabhakar, S. Pankanti, and A. K. Jain, “Biometric Recognition: Security and Privacy Concerns,” *IEEE Security & Privacy*, pp. 33-42, March/April 2003.
[2] Infrared Data Association, http://www.irda.org.
[3] P. Blythe, “RFID for road tolling, road-use pricing and vehicle access control,” *IEE Colloquium on RFID Technology*, pp. 8/1 – 8/6, Oct. 1999.
[4] T. Matsumoto et al., “Impact of artificial gummy fingers on fingerprint systems,” *Proc. SPIE, Optical Security and Counterfeit Deterrence Techniques IV*, vol. 4677, pp. 275-289, Jan. 2002.
[5] Fingerprint Cards, http://www.fingerprints.com.
[6] L. Rila and C. Mitchell, “Security analysis of smartcard to card reader communications for biometric cardholder authentication,” *Proc. USENIX Fifth Smart Card Conference and Advanced Application Conference (CARDIS ’02)*, pp. 19-28, Nov. 2002.
[7] G. Hachez, F. Koeune, and J.-J. Quisquater, “Biometrics, access control, smart cards: a not so simple combination,” *Proc. USENIX Fourth Smart Card Conference and Advanced Application Conference (CARDIS ’00)*, pp. 273-288, Sept. 2000.
[8] A. Noore, “Highly robust biometric smart card design,” *IEEE Transactions on Consumer Electronics,* pp. 1059-1063, Nov. 2000.
[9] S. B. Pam, D. Moon, Y. Gil. D. Ahn, and Y. Chung, “An ultra-low memory fingerprint matching algorithm and its implementation on a 32-bit smart card,” *IEEE Transactions on Consumer Electronics*, pp. 453-459, May 2003.
[10] LEON2 Processor, http://www.gaisler.com.
[11] S. Yang, K. Sakiyama, and I. Verbauwhede, “A compact and efficient fingerprint verification system for embedded systems,” *37th IEEE Asilomar Conference on Signals, Systems, and Computers*, pp. 2058-2062, Nov. 2003.
[12] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley and Sons, 2001.
[13] ATOMIUM Suite, http://www.imec.be/design/atomium.



David D. Hwang received his B.S. and M.S. degrees in Electrical Engineering at University of California, Los Angeles. He is currently pursuing a Ph.D. degree from University of California, Los Angeles in the field of system design and VLSI implementations of secure embedded systems. He is a member of Phi Beta Kappa, Tau Beta Pi, Eta Kappa Nu, and is a graduate fellow of the Fannie and John Hertz Foundation.

Ingrid Verbauwhede (M’92-SM’02) is an Associate Professor of Electrical Engineering at the University of California, Los Angeles. She received the Electrical Engineering Degree and the Ph.D. Degree in electrical engineering from the Katholieke Universiteit Leuven, Belgium in 1991. She was a lecturer and visiting research engineer at UC Berkeley from 1992 to 1994. From 1994 to 1996 she was with TCSI, and from 1996 to 1998 she was with Atmel. She has been on the faculty of UCLA since 1998 and her interests include processor architectures and VLSI design methodologies for real-time, embedded systems in application domains such as cryptography, digital signal processing, algebra, wireless and high speed communications.