Multilevel Design Validation in a Secure Embedded System

Patrick Schaumont, *Senior Member*, *IEEE*, David Hwang, *Member*, *IEEE*, Shenglin Yang, *Student Member*, *IEEE*, and Ingrid Verbauwhede, *Senior Member*, *IEEE*

Abstract—In this paper, we present the simulation-based validation approach that we used during the design of ThumbPod-2, a portable fingerprint authentication system. The particular nature of secure system design has considerable impact on the simulation requirements and design flow. We present two key contributions. We will first show that rigorous design of secure digital systems requires a multilevel validation approach, meaning validation at multiple steps in the design flow. Indeed, an attacker chooses the easiest entry point and does not stick with one abstraction level. Second, we show the use of a cosimulation and codesign environment called GEZEL that can support this type of multilevel validation. We will illustrate this multilevel design validation strategy with the verification of security of the ThumbPod-2 device.

Index Terms—Multilevel simulation, security, embedded systems.

1 Introduction

Modern embedded systems are constituted of heterogeneous hardware and software elements. These elements interact along physical boundaries (buses), software boundaries (function calls), and network boundaries (sockets), both within the device as well as in communication with other agents around them. Validation of such hardware/software devices is a challenging area due to the heterogeneity of simulation environments of the different components.

The development and validation of security for those embedded systems adds another dimension to the validation problem [1]. This is true because not only do the elements of the embedded system require validation, but, in particular, each interface between elements requires extensive validation to ensure security is not compromised. This practice of enforcing security at the boundaries can, for example, also be seen in server-level security coprocessors [2].

In an embedded system, the interface between elements and hardware/software is particularly vulnerable because of their limited physical protection from attackers. A well-known example of a successful attack on an embedded system at the hardware-software interface is the hacking of the X-Box game console [3]. In that system, a secret key used in a software decryption algorithm can be probed from a bus during system bootup. Thus, the cryptographic strength of the software algorithm is completely compromised by a loophole at the hardware level. It shows that embedding security is not a point solution at one abstraction level, but a design approach that covers from the application down to the physical implementation.

Performing simulation-based validation of an embedded system, driven by design-for-security considerations, is the focus of this paper. We rely on simulation techniques 1) because they naturally appeal to a designer interested in implementation and 2) because we are dealing with heterogeneous system implementations, typically consisting of hardware and software.

By itself, the simulation-based validation of mixed hardware-software systems and microarchitectures is a well-explored topic [4]. Recently, complexity issues in embedded system design have called for an increased use of formal tools, next to or in combination with simulation-based approaches [5]. However, security in embedded systems, and the associated validation, is basically different from their design complexity. Increasing the key length of an embedded cipher to strengthen security, for instance, is useless if the key storage is not well protected.

The objective of embedded security is to minimize, both temporally and spatially, the risk of an attack. A secure embedded system contains a root-of-trust [7], a single secret from which all other secrets are derived. The root-of-trust brings security and cryptographic strength to the embedded system, but it also is its Achilles' heel.

The outline of our paper is as follows: In Section 2, we briefly introduce the main design case of this paper: a portable embedded fingerprint authenticator called ThumbPod-2. The biometrics in this application are used as the equivalent of an electronic key. The root-of-trust in ThumbPod-2 consists of a fingerprint minutiae template and a master key. The validation problem in ThumbPod-2 thus needs to demonstrate the systematic protection of those

- *P. Schaumont is with the Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061. E-mail: schaum@vt.edu.*
- *D. Hwang is with KeyEye Communications, Irvine, CA 92618. E-mail: dhwang@ee.ucla.edu.*
- *S. Yang is with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095. E-mail: shengliny@ee.ecla.edu.*
- *I. Verbauwhede is with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095 and the Katholieke Universiteit Leuven, ESAT-COSIC, 3000 Leuven, Belgium. E-mail: ingrid.verbauwhede@esat.kuleuven.be.*

Manuscript received 12 Sept. 2005; accepted 16 Feb. 2006; published online 21 Sept. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-0310-0905.

secrets. In Section 3, we present a methodology for secure system design, based on multilevel system validation. The protection of the root-of-trust can be partitioned into individual design problems at different levels of abstraction (protocol, architecture, microarchitecture, and circuit), and each of these problems can be approached using simulation. We will discuss two validation cases in more detail. One is the design of the fingerprint matching operation and another is the design of the secure protocol. Both of these cases require interaction with the root-of-trust without compromising it. In Section 4, we present GEZEL, the system simulation environment used in the ThumbPod-2 design. GEZEL enables cycle-true cosimulation of hardware and software components and has an integrated path into implementation and stimuli generation. Finally, Section 5 presents the main results obtained out of the ThumbPod-2 design.

2 ThumbPod-2 Embedded Authentication

In this section, we briefly introduce the driver application for our methodology. The ThumbPod-2 system is an embedded authentication system that is able to capture fingerprint images and locally compare the extracted minutiae template to a prestored template. This prestored template belongs to the rightful owner of ThumbPod-2. Successful matching with this template shows that the ThumbPod-2 user and the owner must be the same person; in other words, matching authenticates the identity of the user.

Upon successful matching, the ThumbPod-2 can establish a secure communications link with a server. This link can be used for various electronic transactions that need authentication, such as credit card transactions, personnel access, login operations, remote car locking, and so on.

In contrast to its predecessor, ThumbPod-1 [8], ThumbPod-2 has rigorously applied the principles of secure embedded system design. In particular, care was taken to design and validate a matching protocol that protects the prestored template as a root-of-trust.

Fig. 1 gives an overview of the matching protocol. ThumbPod is a client in a client-server application. Starting with the capture of a user's fingerprint, the device will extract the minutiae out of this fingerprint. These minutiae represent a person-unique signature that can be compared with and matched to a prestored template. Depending on the resulting matching score, ThumbPod will give the user access to a securely stored master key. Meanwhile, the server has generated a random number and communicated this number to ThumbPod. In combination with the master key, ThumbPod can generate a secure session key, valid for the duration of a single successful fingerprint matching. The session key can be used to encrypt a communications payload between the server and ThumbPod. From Fig. 1, we can see that the session key eventually relies on the secrecy of two elements: the master key, which is a shared secret between ThumbPod and the server, and the minutiae template, which is stored only on the ThumbPod. We assume that a secure enrollment protocol has been performed to load the master key and template in the device.

The design problem of ThumbPod-2 is, in short, how do we map the system described above to a combination of software and hardware elements such that we can guarantee the secrecy of the root-of-trust in the resulting implementation? This design objective must be met without assumptions on the possible security attacks on the system. In practice, security breaches will occur at places where one would not expect them (such as the X-box hack mentioned earlier). We found a stepwise, simulation-based multilevel approach particularly effective in implementing our security objective in a systematic manner. This multilevel approach is discussed in the next section.

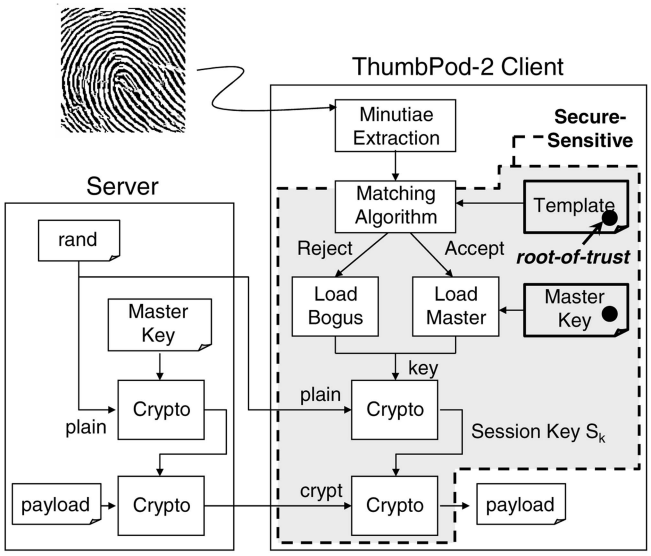

Fig. 1. Security protocol of the ThumbPod-2 client.

3 Multilevel Design Validation of a Secure System

In multilevel design validation, we consider the operation of an embedded system at multiple abstraction levels and target validating each design level by itself. In the context of embedded secure systems, we will validate that the system can withstand security attacks at a single level of design abstraction at a time. Outside of a single level of design, these attacks become side-channel attacks which must be dealt with at a different level of design abstraction.

In a secure embedded system, the security is based on a component that is trusted, which we call the root-of-trust. The root-of-trust is "a component that must always behave in the expected manner, because its misbehavior cannot be detected" [9]. This means that the device behaves as expected even in the presence of adversarial conditions, which are intentional or by accident.

Providing security is a multilayer defense. At the network level, trusted components include certification authorities for public key infrastructure and time servers. At the application level, the system is partitioned into multiple cooperating actors. One of these actors will hold the root-of-trust and embodies it as a secure ID token, a smart-card, a biometric passport. At the architecture level, a single actor is implemented and partitioned between secure software routines—which, for instance, handle biometric data on the token—and nonsecure routines. At the micro-architecture level, the implementation is further refined

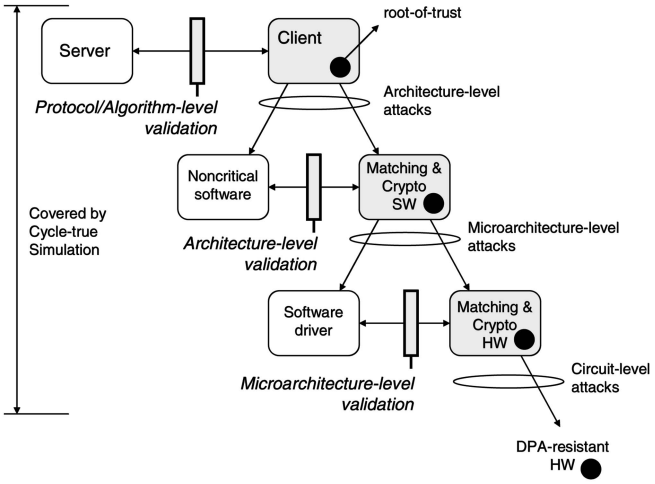

Fig. 2. Multilevel validation of secure embedded systems.

with hardware support, including cryptographic hardware acceleration units and storage units for the sensitive data and keys. At the logic level and circuit level, logic styles and design methods are proposed to make the hardware resistant to side channel leakage in the implementation in the form of power and timing attacks.

The main reasons why the complete system cannot be protected are cost and complexity. Security measures have a cost associated that one wants to minimize. In addition, a secure component should minimize complexity and be as small as possible as it eases the protecting and monitoring of the system boundaries and interfaces.

Especially important in the context of multilevel design validation is a clear and well-defined security boundary. It is the interface between the trusted or secure part and the untrusted, regular part of the embedded system. The security boundary exists at different levels of abstraction. At the application level, the security boundary is at a trusted server or at an authority that issues public key certificates. In the context of embedded systems, the security boundary consists of transactions, instructions, buses, registers, and physical wires. The recursive multilevel security partitioning and associated multilevel validation for a typical secure embedded system is shown in Fig. 2.

At the protocol level, the entire embedded system is considered trusted because it holds a root-of-trust. The validation of the communication protocol thus targets showing that this root-of-trust cannot leak by illegal protocol sequences. This model applies to a large class of secure embedded systems, such as secure tokens, smartcards, electronic ID cards, etc. The embedded device itself can still be attacked at the architecture level by interfering with the implementation of the embedded software and hardware inside of the device. These attacks are thwarted by partitioning the device at the architecture level in a trusted and a nontrusted part, which will isolate the root-of-trust to a confined area. In Fig. 1, this area is indicated as a secure-sensitive shaded area. A typical embedded system will contain one or multiple, usually very heterogeneous, processor cores. Some of these cores or coprocessors will execute secure operations, while others will be running regular software. Similarly, there will be a partioning between secure and nonsecure storage. Very sensitive parts of the device need to be designed in tamper proof and/or side-channel-resistant environments.

Multilevel validation thus recognizes that the protection of a root-of-trust is a recursive problem. In order to consider all possible side-channel attacks, it is necessary to consider system operation at multiple abstraction levels and to validate that the security assumptions of the higher levels also hold up at the lower levels.

In the following paragraphs, we give examples of validation at different abstraction levels in the ThumbPod-2 design. We first briefly introduce the design and validation environment that we have used in solving this security partitioning problem.

3.1 Cycle-True Platform Validation in GEZEL

The target architecture of a typical secure embedded device consists of an embedded processor core, to implement the software parts, combined with one or more secure hardware accelerators. In our case, these hardware accelerators are developed using special circuit-level techniques that protect the accelerator against circuit-level side-channel attacks [10]. The ThumbPod-2 is an illustration of this target architecture.

In this paper, we will focus on the validation techniques needed at the protocol/algorithm, architecture, and micro-architecture levels and assume that the circuit level will be side-channel attack resistant by construction.

Following the stepwise approach discussed in Fig. 2, we target using simulation at multiple abstraction levels, including the functional level, as well as architecture-level simulations. Table 1 lists the simulation and modeling accuracies required for each of the design levels. We use the

TABLE 1
System Design Abstraction Levels for Secure Embedded Systems

Security abstraction level	Minimum Simulation Accuracy	Modeling accuracy
Application Protocol	Protocol Transactions	Actor model
Algorithm	Function accurate	Behavioral model
Architecture	Instruction accurate	Instruction-set model
Micro-Architecture	Cycle accurate	Hardware model
Circuit	Continous-time	Device model

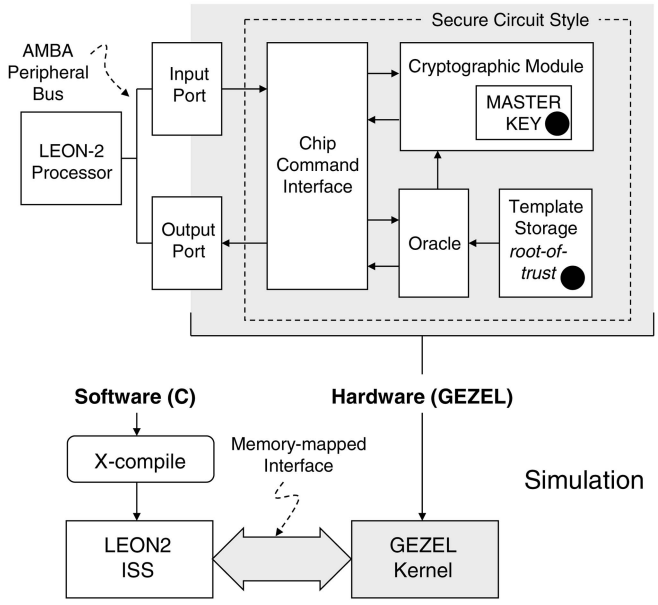

Fig. 3. Thumbpod-2 Platform (top) and system simulation (bottom).

GEZEL design environment, which captures cycle-true descriptions of hardware and which also supports cosimulation of those hardware descriptions with embedded software. Using GEZEL, we support security partitioning and the associated multilevel design validation required for the shaded area of Table 1. We thus assume here that we have a side-channel-free circuit technology available that can serve as a target for the secure part of GEZEL models. An example of such a technology is WDDL [11]. Rather then using the minimum simulation accuracy required for each level, we choose to consistently use cycle-true simulation throughout the design. This higher simulation accuracy benefits the investigation of side-channels based on timing and power consumption. The power estimation model uses Hamming-distance toggle counting, where, optionally, a distinction can be made between upgoing transitions (which, in CMOS, drain current) and downgoing transitions (which, in CMOS, source a current).

The bottom part of Fig. 3 shows how the ThumbPod-2 platform can be modeled as the combination of an embedded core and a coprocessor in secure circuit technology. The coprocessor includes a cryptographic module and an oracle. Both of these will be discussed further. The GEZEL kernel [12] provides simulation support for the hardware part of the design, while an instruction-set simulator for LEON2 is used to support the software part. The system simulation is cycle-accurate. The cosimulation speed of the complete platform using GEZEL and the LEON2 ISS is about 50KHz on a 3GHz Pentium PC when the system is fully exercised.

The communication between the embedded core and the ThumbPod-2 is done by means of memory-mapped interfaces and GEZEL provides modeling constructs for these interfaces next to the hardware modeling for the coprocessor. When the GEZEL coprocessor is part of the side-channel attack resistant implementation, the memory-mapped interfaces will become the boundary between the secure and the nonsecure part of the system. The validation of the security properties at different abstraction levels in the system can then focus on these interfaces.

- At the architecture-level, one can simulate various logical out-of-sequence accesses to the coprocessor in GEZEL. These exceptional sequences can usually be derived out of the higher-level security protocols and an example of such a sequence will be provided in Sections 3.2 and 3.3.
- At the microarchitecture level, one can look at the cycle-true timing behavior of memory reads and writes and investigate possible side-channels based on timing or simple power-analysis. A well-known example of this can be found in elliptic-curve encryption (ECC) of public keys, where the key-bits translate directly to a different sequence of coprocessor instructions and where they, consequently, can be picked up easily [13].

Besides memory-mapped interfaces for processors, GEZEL also supports other interfaces depending on the kind of processor or host system that is being used for the nonsecure part of the system. For example, we have used an 8051 microcontroller with a port-mapped interface to design secure public-key coprocessors for smartcards. Another example is the use of the coprocessor interface on an ARM processor to obtain a performance-optimized secure coprocessor implementation.

Finally, security can also be implemented at the physical layer by means of tamperproof hardware and side-channel resistant circuit styles; though these implementations are beyond the scope of this paper, the same security design principles can be applied at these lowest levels [11].

In the following, we will discuss how the ThumbPod-2 security protocol shown in Fig. 1 maps into the platform of Fig. 3 and how the cosimulator helps in system validation.

3.2 Protocol Transformation

In this section, we discuss the transformation of a secure protocol from a server-client model to a server-insecure software client-secure hardware architecture model. This transformation must be done with validation at each new interface introduced to prevent leakage of information from the root-of-trust.

Consider the biometric verification protocol as shown in Fig. 4. In this figure, a server and a device share a secret master key K. The server also stores a secure one-way hash of the user's master fingerprint template, which is nonsensitive due to the noninvertible nature of the keyed-hash function. The device stores the actual fingerprint template. The protocol begins by the server generating a pair of random numbers (RAND, RANDT) and forwarding these to the device. The device performs a live biometric feature extraction and match on the candidate fingerprint. If a match is made, the device creates a session key SK using RAND and the master key K. It also creates a keyed hash of the template (to compare with the server's stored version). The device generates two authentication tokens based on the session key, the random value RANDT, and the keyed hash of the template. If any part of the protocol is incorrect, the device generates dummy tokens.

VERIFICATION PROTOCOL (CHRONOLOGICAL ORDER)

Device → Server	ID
Server	Load corresponding K
	Generate RAND, RANDT, SK = E(K, RAND)
Server → Device	RAND | RANDT
Device	Obtain RAW IMAGE from user
	Perform FEATURE EXTRACTION algorithm to obtain MINUTIAE
	Load TEMPLATE
	Perform MATCH algorithm of MINUTIAE versus TEMPLATE
	If match, load K and generate HASH = H(K,TEMPLATE)
	Else, set K = 0000 and set HASH = 0000
	Generate SK = E(K,RAND)
	Generate TOKEN1 = E(SK,RANDT) and TOKEN2 = E(SK,HASH)
Device → Server	TOKEN1 | TOKEN2
Server	Decrypt TOKEN1 | TOKEN 2 to verify device and interpret result.
Server → Device	TRANSACTION RESULT

Fig. 4. Original biometric protocol.

A protocol running on an unprotected embedded processor, however, is vulnerable to a number of architecture and software attacks. To improve security, a protocol transformation is required to move all sensitive variables and functions in a secure hardware root-of-trust.

To perform the transformation, the designer must first determine which data elements must be contained within the root-of-trust and which can be left exposed. The variables in the root-of-trust are called variables of trust. In the protocol in Fig. 4, the master key K and the template must remain in the root-of-trust.

Once data variables of trust are determined, the next step in the transformation is to determine which functions directly use the variables of trust in a way that could leak sensitive information. In our case, the matching algorithm, the keyed-hash generation, the session key generation, and token generation are such functions. Thus, these functions of trust must be mapped into the root-of-trust to prevent leakage out of the root-of-trust.

After determining the variables and functions of trust (shown in bold in Fig. 4), the next transformation step is to partition these elements onto the hardware root-of-trust, leaving the other elements in insecure software. Finally, a communication protocol must be constructed such that the root-of-trust can communicate securely with the insecure software. In our case, this leads to the enhanced biometric protocol shown in Fig. 5.

In terms of microarchitecture, the root-of-trust is implemented as a memory-mapped coprocessor. We interface the hardware coprocessor to an insecure processor via a memory-mapped interface with a 16-b instruction bus (INS), 32-b input data bus (DIN), and 16-b output data bus (DOUT).

A instruction set must be designed to interface between the insecure software and the hardware coprocessor. The only communication between the insecure coprocessor and the hardware coprocessor (root-of-trust) at the architecture-level is these three buses. Hence, the communication across these buses and their ensuing security is what must be validated.

The instruction set between software and secure coprocessor includes 38 instructions, 11 of which support the mapping the biometric protocol (Fig. 5), 11 others support the matching oracle (discussed further), and 16 others implement debug and BIST mechanisms on the coprocessor microarchitecture.

The biometric protocol maps each atomic action of the protocol into an instruction. For example, there is an instruction to send the RAND-RANDT values from the insecure coprocessor to the secure processor. After the passing of the data, another $\$\mathrm{begin\_encrypt}$ instruction allows the secure coprocessor to generate the first token (or a dummy token). After the token is generated and therefore consumed by the processor, the processor sends a $\$\mathrm{do\_hash}$ instruction to generate the second token. After receiving the second token, the insecure processor is able to send this to the server for final verification.

Internal to the hardware coprocessor are a number of security mechanisms, such as security flags and sequencing counters, to protect the coprocessor from false instruction and out-of-sequence instruction attacks.

We will now focus on a specific sequence of the biometric protocol, namely, the biometric matching. The location of the biometric matching in the overall biometric protocol of Fig. 5 is marked with a double sidebar.

3.3 Biometric Matching Oracle

The biometric matching process compares the minutia set of a user to a template minutia set. This template is prestored and belongs to the true owner of the ThumbPod. The original matching algorithm of the ThumbPod is based on pairwise comparison of the minutia set and is developed as an algorithm in C. The algorithm relies on calculations in polar space and may be consulted in [14]. The validation of this algorithm in C requires simulation against a database of sample fingerprints to establish reliability bounds (false-accept and false-reject ratios, as described in [14]). The C code is also run on top of the instruction-set simulator to evaluate the performance on the target embedded processor.

For the security point-of-view, the complete C algorithm requires the highest level of trust because it directly manipulates the template minutia (the root-of-trust). Subsequent refinements of the algorithm therefore target isolating the root-of-trust to a confined area of the implementation, which will eventually map into a secure circuit style.

At the architecture level, we therefore introduce the concept of an oracle. The oracle will hide the matching

VERIFICATION PROTOCOL (CHRONOLOGICAL ORDER)

Device (I) → Server	ID
Server	Load corresponding K
	Generate RAND, RANDT, SK = E(K, RAND)
Server → Device (I)	RAND | RANDT
Device (I)	Obtain RAW IMAGE from user
	Perform FEATURE EXTRACTION algorithm to obtain MINUTIAE
Device (I) → Device (S)	**NUM_MINUTIAE**
	{j | kk | MINUTIA} x (NUM_MINUTIAE (i)
	x NUM_TEMPLATE (j) x NUM_NEI_MIN (k) x NUM_NEI_TEMPL (kk))
Device (S)	**Store NUM_MINUTIAE**
	For each MINUTIA received {
	Load template region for TEMPLATE j and TEMPLATE NEIGHBOR kk
	Local MATCH algorithm with MINUTIA i and MINUTIA NEIGHBOR k
	Update partial match registers
	}
	After all MINUTIAE, obtain final match result
	If match, set MATCH_FLAG = 1; Else, set MATCH_FLAG = 0
Device (I) → Device (S)	**RAND | RANDT**
Device (S)	**If TEMPL_LD_FLAG == 1 & SEQUENCE == 4 {**
	Store RAND and RANDT
	Set MATCH_FLAG_CRYPTO = MATCH_FLAG and SEQUENCE = 12
	}
Device (I) → Device (S)	**BEGIN_ENCRYPT**
Device (S)	**Set MATCH_FLAG = 0**
	If TEMPL_LD_FLAG == 1 & SEQUENCE == 12 {
	If MATCH_FLAG_CRYPTO == 1, load K; Else, set K = 0x0000
	Generate SK = E(K,RAND)
	Generate TOKEN1 = E(SK,RANDT) and set SEQUENCE = 13
	}
	Else, set TOKEN1 = 0x0000
Device (S) → Device (I)	**TOKEN1**
Device (I) → Device (S)	**DO_HASH**
Device (S)	**If TEMPL_LD_FLAG == 1 & SEQUENCE == 13 {**
	If MATCH_FLAG_CRYPTO == 1, generate HASH = H(K,TEMPLATE);
	Else, set HASH = 0x0000
	Generate TOKEN2 = E(SK, HASH)
	Set MATCH_FLAG_CRYPTO = 0 and SEQUENCE = 4
	}
	Else, set TOKEN2 = 0x0000 and MATCH_FLAG_CRYPTO = 0
Device (S) → Device (I)	**TOKEN2**
Device (I) → Server	TOKEN1 | TOKEN2
Server	Decrypt TOKEN1 | TOKEN 2 to verify device and interpret result
Server → Device (I)	TRANSACTION RESULT

Fig. 5. Transformed biometric protocol.

operations that lead up to the accept/reject decision for a users' fingerprint, and only announce the final result, similar to the oracle concept in Greek Mythology. The oracle must remain sufficiently simple because it will eventually map into hardware. We rewrote the C code for the original algorithm so that the oracle was clearly identifiable in the complete matching algorithm as a separate set of functions. Doing these transformations at the C level enabled us to use a single fast instruction-set simulator throughout the architecture validation phase. The result of this partitioning algorithm is that the minutia template, combined with the comparison operations that touch it, are isolated as a root-of-trust in the program.

In the third step of the refinement, the microarchitecture for the oracle was created as a GEZEL module that was cosimulated with a driver program in C. The driver program presents the same API as the architecture-level oracle, but, in fact, interfaces to a memory-mapped hardware implementation of this oracle. The GEZEL description is a cycle-true, register-transfer level description of the protocol architecture which can implement all operations of the oracle by means of a custom instruction set, as was discussed earlier, in Section 3.2. Table 2 shows the example of the 11 instructions that were created to control the oracle microarchitecture.

In the next section, we turn to the issue of microarchitecture modeling in GEZEL and cosimulation of the coprocessor with C.

4 Coprocessor Modeling, Implementation, and Verification in GEZEL

In the previous two sections, we have described a partitioning process that isolates the root-of-trust over multiple abstraction levels. At the microarchitecture level, this partitioning results in custom hardware architectures which are described in the GEZEL language and which are then cosimulated with C running on an instruction-set simulator. In this section, we will briefly describe the GEZEL modeling language and discuss the implementation support offered by a GEZEL-based environment. This support consists of 1) automatic generation of synthesizable VHDL and 2) generation of register transfer (RT) level and chip-level test-vector stimuli.

TABLE 2
System Design Abstraction Levels for Secure Embedded Systems

Op	Mnemonic	Operation
84	ins_oracle_start	Reset oracle and prepare for matching
85	ins_oracle_index	Instruction storing which minutiae (0 through 29) to compare against.
86	ins_oracle_index_nei	Instruction storing which neighbor (0 to 5) to compare against.
87	ins_oracle_dis_input	Store DIN as candidate DIS_IN for matching.
88	ins_oracle_phi_input	Store DIN as candidate PHI_IN for matching.
89	ins_oracle_sita_nei_input	Store DIN as candidate SITA_NEI_IN for matching.
90	ins_oracle_sita_input	Store DIN as candidate SITA_IN for matching.
94	ins_oracle_load	Load the first minutia into template storage.
95	ins_oracle_load_last	Load the last minutia into template storage.
98	ins_oracle_loadtemplatenum	Load the number of template minutiae (NUM_TEMPL) into the oracle and crypto engine.
99	ins_oracle_loadinputnum	Load the number of candidate minutiae (NUM_MINUTIAE) into the oracle.

4.1 Microarchitecture GEZEL Models for Codesign

Let us consider again the secure_match() operation of the biometric protocol. As can be seen in the architecture validation step of Fig. 6, this step tests the value of variable match_count against a preset value N. match_count holds the number of minutia in the input fingerprint that are considered matching (equal) to those in the template fingerprint. When this number exceeds the preset value N, the input fingerprint is considered similar to the template. The actual value of match_count is, however, secure-sensitive since it has an obvious relationship to the prestored template minutia (root-of-trust). For example, an attacker could exhaustively traverse the input template minutia space with artificially generated minutia and extract the secret template by observing the value of match_count during the traversal.

Therefore, at the microarchitecture level, the function secure_match() is separated into a secure hardware part described in GEZEL and a software driver in C. Listing 1 and Listing 2 illustrate a (simplified) model of the GEZEL and C part of this model.

GEZEL is a cycle-true register-transfer modeling language. A system consists of a number of modules, each of which are expressed as the combination of a data path (dp, as in lines 1, 6, and 33) and a controller (hardwired, line 5, or fsm, line 16). A GEZEL data path can have one or more instructions or sfg, each of which will take a single clock cycle to execute. The controller of a data path will define an execution schedule for those sfg.

The module matchmod, in lines 1-5 in Listing 1, holds a single register matchcnt which will hold the secret

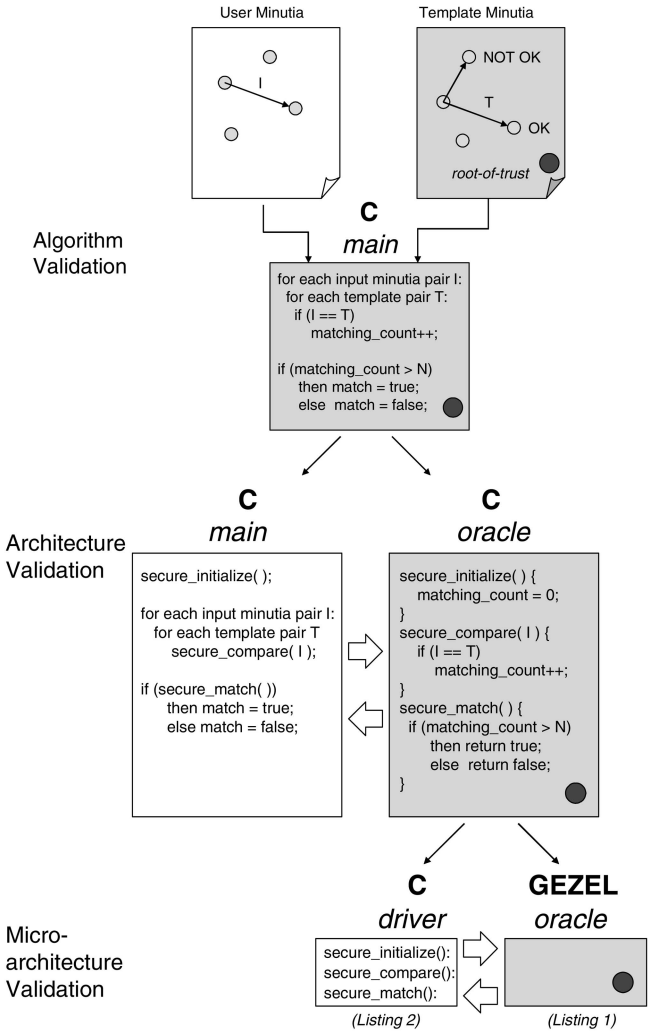

Fig. 6. Multilevel design validation of the biometric oracle.

number of matched minutia. The module also has a 1-bit output port match that will provide a flag representing the result of the threshold comparison for match_count. A hardwired controller, as in line 5, executes the same instruction each clock cycle. The matchmod module in lines 1-5 is a strongly simplified version of the actual oracle. It assumes that matchcnt will be generated by some unspecified GEZEL logic.

LISTING 1. GEZEL module for `secure_match`

```
1. dp matchmod(out match : ns(1)) {
2.    reg matchcnt : ns(5);
3.    sfg compare { match = (matchcnt > 20); }
4. }
5. hardwired h_match_mod(matchmod) {compare;}
6. dp matchmod_decoder(in ins     : ns(8);
7.                     out dout   : ns(32) ) {
8.    sig match      : ns(1);
9.    reg ireg       : ns(8);
10.    reg doutreg   : ns(32);
11.    use match_module(match);
12.    sfg dec      { ireg = ins;
13.                   dout = doutreg;}
14.    sfg rmatch   { doutreg = match; }
```

```
15. }
16. fsm h_matchmod_decoder(matchmod_
                          decoder){
17.    initial s0;
18.    state s1, s2;
19.    @s0 (dec) -> s1;
20.    @s1 if (ireg==1) then (dec, rmatch)
                                      -> s2;
21.        else (dec)                -> s1;
22.    @s2 if (ireg==0)} then (dec)  -> s1;
23.        else (dec)                -> s2;
24. }
25. ipblock b_ins}(out data : ns(8)) {
26.    iptype ''leonsimsource'';
27.    ipparm ''address=0×20000000'';
28. }
29. ipblock b_dataout(in data : ns(32)) {
30.    iptype ''leonsimsink'';
31.    ipparm ''address=0×20000004'';
32. }
33. dp sysmatch {
34.    sig ins   : ns( 8);
35.    sig dout  : ns(32);
36.    use match_module_decoder(ins, dout);
37.    use b_ins(ins);
38.    use b_dataout(dout);
39. }
40. system S {
41.    sysmatch;
42. }
```

LISTING 2. C driver for `secure_match`

```
1. #include <stdio.h>
2.
3. enum {ins_idle, ins_readmatch};
4. int secure_match() {
5.      volatile unsigned char *ins
6.      = (volatile unsigned char *)
                  0×20000000;
7.    volatile unsigned int *dout
8.      = (volatile unsigned int *)
                  0×20000004;
9.    *ins = ins_readmatch;
10.    rv = *dout;
11.    *ins = ins_idle;
12.    return rv;
13. }
```

The second module in Listing 1 (lines 6-24) represents an instruction decoder that interfaces the C driver on the embedded processor to matchmod. Such an instruction-set decoder multiplexes the memory bus over the different hardware I/O ports that need to be accessed and synchronizes the embedded software to the secure hardware module. The instruction decoder illustrates the use of structural hierarchy (line 11) and the use of a finite-state-machine specification to express conditional sfg sequencing (lines 16-24). The instruction decoder is connected to the embedded software by means of memory-mapped interfaces (lines 25-32), which specify the shared address location between a hardware port at GEZEL level (e.g., port data on line 25) and a memory address for the C program (e.g., address 0×20000000 on line 27).

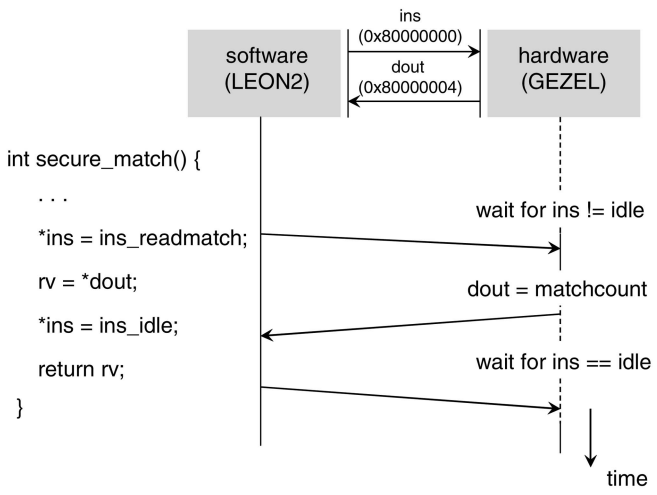

Fig. 7. One-way handshaking between C and GEZEL.

A C driver for this GEZEL module is shown in Listing 2. As was shown earlier in Fig. 6, the objective of this driver is to map a C API (secure_match()) into accesses to a secure hardware module. Because of the memory-mapped interfaces, these accesses can be represented using memory read/writes on initialized pointers (Listing 2, lines 5-8). As can be seen from the driver C description, the value of the secure match comparison is obtained as a single memory read (line 10). Detailed security aspects, such as how many minutia in the input actually matched to the template (the value of match-count in Listing 1), remain private to the hardware module. The combination of the software in Listing 2 and the hardware in Listing 1 results in a one-way handshake protocol, as illustrated in Fig. 7. This one-way handshake protocol assumes that the hardware needs to synchronize the software but not vice versa.

4.2 Implementation of GEZEL Modules

Besides cosimulation, GEZEL also supports a path to implementation. This support consists of VHDL code generation, as well as the possibility of recording block-level I/O stimuli as well as chip-level I/O stimuli. The VHDL is generated at the RTL level. In contrast to typical transaction-level C-based cosimulation models, GEZEL models are fully convertible to VHDL. That is, if a module can be written and simulated in GEZEL, then there exists a direct translation of this module into synthesizable VHDL.

Listing 3 shows an example of how I/O stimuli are recorded during GEZEL cosimulation. Using a trace directive in I/O signals (lines 5 and 6), interface signals are probed and placed into a text file with one value written per cycle. These stimuli can be reused for RT-simulation of the VHDL and, later, for chip-level validation of the actual chip. RT-simulation at VHDL of the complete system is clearly out of the question. However, we did simulations at the individual block level, primarily to verify the correctness of the generated VHDL code.

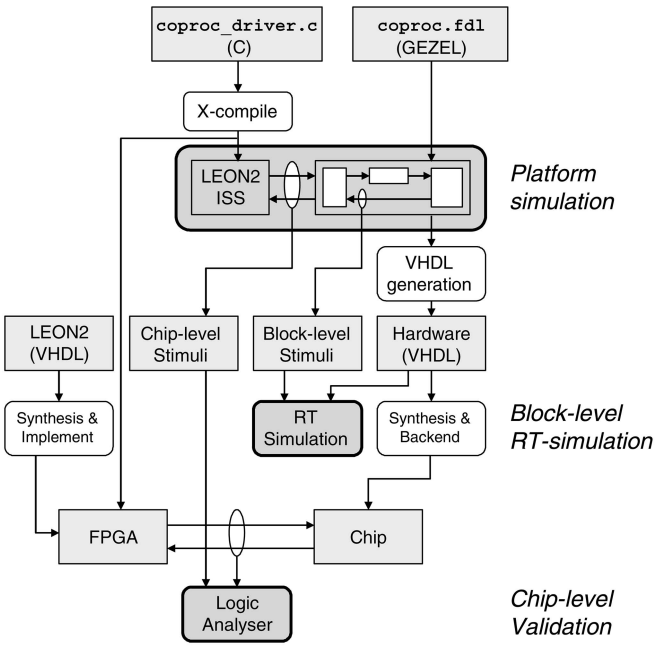

Fig. 8. Design flow with back-end validation levels.

LISTING 3. Stimuli trace recorders in GEZEL

```
1. ...
2. dp sysmatch {
3.    sig ins   : ns( 8);
4.    sig dout  : ns(32);
5.    $trace(ins, vectors/ins.txt);
6.    $trace(dout, vectors/out.txt);
7.    use match_module_decoder(ins, dout);
8.    use b_ins(ins);
9.    use b_dataout(dout);
10. }
```

5 Results

In this section, we summarize the validation and implementation results of the complete ThumbPod-2 system.

5.1 Validation of the Final Protocol

The protocol between the insecure processor and the secure coprocessor that holds the root-of-trust is fully validated at the microarchitecture level. Using a GEZEL model for the coprocessor and its interface and an ISS of the LEON processor, an embedded C test program was written to verify the proper protocols of the system and test a number of potential attacks.

The C testbench coproc_driver.c was simulated on the instruction-set simulator and it passed instructions and data to/from the GEZEL coprocessor. It tested all 38 instructions in various orders for a total of over 230,000 instructions tested. The test bench verified the normal operation of the protocol and correctly withstood various forms of out of sequence and improper instruction attacks. A software-coded built-in self-test (BIST) was also implemented to test the encryption modes of the final device.

5.2 Implementation and Design Validation Flow

The implementation flow for the ThumbPod-2 system is shown in Fig. 8. After platform cosimulation using C and GEZEL, the GEZEL modules are converted into synthesizable RTL-VHDL and test-bench stimuli are recorded into files. These stimuli are used for block-level RT simulations in VHDL. LEON's output to the coprocessor (instruction and input data buses) was modeled by the appropriate stimuli files. The coprocessors' return path to the LEON (output bus) was written into another test file and automatically tested versus the known original output file. Hence, RTL validation of the coprocessor could be performed without RTL simulation of the LEON processor. The RTL-validated VHDL is synthesized into the final ASIC implementation. The LEON2 processor was not implemented on the same

TABLE 3
Coding and Performance for the Biometric Oracle

Level	C NCLOC*	GEZEL NCLOC	Performance (cycles)
Algorithm	311		188.2M
Architecture	331		188.4M
Micro-Architecture	321	255	73.0M

** NCLOC = Non-Comment Lines of Code*

TABLE 4
Code Lengths of the Coprocessor

Module	GEZEL NCLOC	RTL VHDL NCLOC	Gate-level VHDL NCLOC
Interface	154	402	573
Oracle	255	3517	31,690
Crypto	364	1623	4,804
Memory	—	2574	30,004

TABLE 5
Area of the Coprocessor

Module	Area (mm^2)	Eq. gates
Interface	0.023	2 K
Oracle	0.114	11 K
Crypto	0.794	79 K
Memory	1.054	105 K

die, but separately implemented on an FPGA. When the chip returned from fabrication and packaging, a test setup was created that combined an FPGA board with a test board holding the ASIC. This way, chip testing could use the same coproc_driver.c testbench that was used during system-level conception of the system. As a result, the GEZEL environment allowed smooth validation of the ISS model, the RTL model, the gate-level VHDL model, and the final fabricated IC.

5.3 Coding and Design Complexity

Finally, we document the coding and design complexity of the design in Table 3, Table 4, and Table 5. Table 3 illustrates how the design size of the biometric oracle evolves over the different modeling abstraction levels.

The final fabricated IC consisted of four components:

1. the interface between the coprocessor and processor,
2. the oracle,
3. the cryptographic engine based on AES, and
4. the storage memory of the fingerprint.

Table 4 shows the size of the design descriptions and the implementation for each of those components.

6 Conclusions

System-level design of secure embedded system such as ThumbPod-2 includes multiple levels of design. These levels each take care of specific security and integrity aspects of the system and each of those can be modeled as a separate validation problem. For validation at the micro-architecture level, we used a codesign environment called GEZEL to perform cycle-true system-level validation. The IC that was fabricated in this methodology has been proven operational and, moreover, was shown to be able to withstand a wide range of security attacks.

Acknowledgments

The authors acknowledge the support of the Fannie and John Hertz Foundation, US National Science Foundation, SRC, and FWO.

References

[1] S. Ravi, "Security in Embedded Systems: Design Challenges," *ACM Trans. Embedded Computing Systems,* special issue on security, vol. 3, no. 3, pp. 461-491, Aug. 2004.
[2] T. Arnold and L.P. Van Doorn, "The IBM PCIXCC: A New Cryptographic Coprocessor for the IBM eServer," *IBM J. Research and Development,* vol. 48, nos. 3/4, pp. 491-503, 2004.
[3] A. Huang, "Keeping Secrets in Hardware: The Microsoft XBox Case Study," AI Memo 2002-008, Massachusetts Inst. of Technology 2002.
[4] S. Edwards, L. Lavagno, E. Lee, and A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis," *Proc. IEEE,* vol. 85, no. 3, pp. 366-390, Mar. 1997.
[5] P. Mishra, N. Dutt, N. Krishnamurthy, and M. Abadir, "A Top-Down Methodology for Microprocessor Validation," *IEEE Design and Test of Computers,* vol. 21, no. 2, pp. 122-131, Mar.-Apr. 2004.
[6] S.W. Smith, R. Perez, S.H. Weingart, and V. Austel, "Validating a High-Performance, Programmable Secure Coprocessor," *Proc. 22nd Nat'l Information Systems Security Conf.,* Oct. 1999.
[7] S. Pearson, "Trusted Computing Platforms, the Next Security Solution," HP Technical Report HPL-2002-221, Nov. 2002.
[8] P. Schaumont, K. Sakiyama, Y. Fan, D. Hwang, B. Lai, A. Hodjat, S. Yang, and I. Verbauwhede, "Testing ThumbPod: Softcore Bugs Are Hard to Find," *Proc. IEEE Int'l High Level Design Validation and Test Workshop (HLDVT '03),* pp. 77-82, Nov. 2003.
[9] Trusted Computing Group, http://www.trustedcomputinggroup.org, 2006.
[10] K. Tiri, D. Hwang, A. Hodjat, B. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, "A Side-Channel Leakage-Free Co-Processor IC in 0.18 um CMOS for Embedded AES-Based Cryptographic and Biometric Processing," *Proc. Design Automation Conf.,* pp. 222-227, June 2005.
[11] K. Tiri and I. Verbauwhede, "Simulation Models for Side-Channel Information Leaks," *Proc. 2005 Design Automation Conf.,* pp. 228-233, June 2005.
[12] The GEZEL Design Environment, http://rijndael.ece.vt.edu/gezel2, 2006.
[13] K. Sakiyama, L. Batina, P. Schaumont, and I. Verbauwhede, "HW/SW Co-Design of TA/SPA-Resistant Public-Key Crypto-Systems," *Proc. Workshop Cryptographic Advances in Secure Hardware,* Sept. 2005.
[14] S. Yang and I. Verbauwhede, "A Secure Fingerprint Matching Technique," *Proc. ACM Workshop Biometrics: Methods and Applications,* pp. 89-94, Nov. 2003.

Patrick Schaumont received the MS degree in computer science from Rijksuniversiteit Ghent, Belgium, and the PhD degree in electrical engineering from the University of California, Los Angeles (UCLA) in 1990 and 2004, respectively. He is an assistant professor in the Electrical and Computer Engineering Department of Virginia Tech. Before joining UCLA in 2001, he had been a researcher at IMEC, Belgium, since 1992. His research focuses on design methods and architectures for embedded systems and he works in close cooperation with designers to demonstrate new methodologies on practical applications. He is a senior member of the IEEE.

David Hwang received the MS degree from the University of California, Los Angeles (UCLA) in December 2001, researching architectures and ASIC implementations of VLSI digital signal processing systems. He received the PhD degree in electrical engineering from UCLA in March 2005. His research focused on VLSI implementations and architectures for cryptographic and secure systems. He is currently with KeyEye Communications investigating DSP architectures for multi-gigabit Ethernet transceivers. He was a UC Regents Scholar, a Department of Defense NDSEG Fellow from 1999 to 2000, and a Hertz Foundation Graduate Fellow from 2000 to 2005. He is a member of the IEEE.

Shenglin Yang received the BS and MS degrees in electronics from Beijing University, Beijing, China, in 1998 and 2001, respectively. She is currently pursuing the PhD degree in electrical engineering at the University of California, Los Angeles. Her research interests include biometrics, pattern recognition, embedded systems, and security. She is a student member of the IEEE.

Ingrid Verbauwhede received the electrical engineering degree in 1984 and the PhD degree in applied sciences from the Katholieke Universiteit Leuven (K.U. Leuven), Belgium, in 1991. She was a lecturer and visiting research engineer at the University of California (UC), Berkeley, from 1992 to 1994. From 1994 to 1998, she was a principal engineer, first with TCSI and then with Atmel in Berkeley, California. She joined the University of California, Los Angeles (UCLA), in 1998 as an associate professor and joined the K.U. Leuven in 2003. At K.U. Leuven, she is codirector of the ESAT-COSIC research group. Her research interests include circuits, processor architectures and design methodologies for real-time, embedded systems in application domains such as security, cryptography, digital signal processing, and wireless applications. She was the general chair of the IEEE International Symposium on Low Power Electronic Devices (ISLPED) in 2003. She is or has been a member of several program committees, including DAC, ISSCC, DATE, CHES, ICASSP, SIPS, and ASAP. She is the design community chair for the 42nd and 43rd DAC executive community. She is a senior member of the IEEE.