Multi-level Design Validation in a Secure Embedded System

David D. Hwang
UCLA Elec. Eng. Dept.
dhwang@ee.ucla.edu

Patrick Schaumont
Virginia Tech ECE Dept.
schaum@vt.edu

Shenglin Yang
UCLA Elec. Eng. Dept.
shengliny@ee.ucla.edu

Ingrid Verbauwhede
UCLA Elec. Eng. Dept.
K.U. Leuven
ingrid@ee.ucla.edu

***Abstract*—We present a simulation-based methodology to support secure embedded design. The methodology is explained through a case study, the Thumbpod-2 portable embedded fingerprint authenticator. By using multi-level validation, we can observe the flow of sensitive information through the system as it takes on multiple forms, from software variables to hardware bus-signals. This allows shielding off of unwanted side-channel information leaks at the protocol, software, or hardware level. We discuss how the ThumbPod-2 design is partitioned into a side-channel-free implementation, and how a codesign environment called GEZEL is used to validate this partitioning process at each abstraction level.**

I. INTRODUCTION

Modern embedded systems are constituted of heterogeneous hardware and software elements. These elements interact along physical boundaries (buses), software boundaries (instruction 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 boot-up. Thus, the cryptographic strength of the software algorithm is completely compromised by a loophole at the hardware level.

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 (a) because they naturally appeal to a designer interested in implementation and (b) because we are dealing with heterogeneous system implementations, typically consisting of hardware and software.

By itself, simulation-based validation hardware-software codesigned systems and micro-architectures 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.

Side-channel attacks are the collection of techniques that identify and exploit information leaks. A basic characteristic of these attacks is that they bypass or violate the assumptions that have been made on the security protection of a certain device. Side-channel attacks can be classified according to the their operating abstraction level, such as illustrated by the security pyramid of Table I [6]. The security pyramid contains five levels, ranging from abstract security protocols to detailed circuit-level implementation.

TABLE I. SECURITY PYRAMID AND SIDE-CHANNEL ATTACKS.

Security Abstraction Level	Security Objective Example	Typical Embedded Implementation Target	Side-Channel Attack Example
Security Protocol	authentication, confidentiality	software	man-in-the-middle
Algorithm	encryption, hashing	software	known-plaintext, known-ciphertext
Architecture	functional integration (SW)	software	software-bypass
Micro-Architecture	architecture integration (HW)	hardware	bus-probing
Circuit	implementation	hardware	differential power-analysis

Security countermeasures protect each level of the security pyramid separately. For example, by choosing a longer encryption key, the algorithm level can be safeguarded against known-plaintext and known-ciphertext attacks. But at the same time, each level of the security pyramid is also subject to side-channel attacks, and typically any abstrac-

tion level can be used to attack another one. For example, increasing the keylength of an encryption algorithm is useless if the key value can be probed electrically from the micro-architecture [3], or if the security protocol fails to maintain secrecy while distributing the key among partners (man-in-the-middle attack).

Various techniques are available for defense against side-channel attacks, including tamper-proofing, keeping trusted and non-trusted computing parts separated by identifying and implementing a set of trusted operations [7], and carefully selecting and designing the crypto-algorithm and protocol. There is also a wide range of techniques available to thwart power-analysis attacks. In general, these techniques target making the power consumption pattern independent of the actual computations done by the design. At the gate level, specialized circuit styles exist to give individual gates a quasi-data-independent power dissipation [9]. At the macro level, a macro can be replaced by two macros, with completely different power consumption profiles, which are randomly inserted into the datapath. Using masking, the information flow through a cipher is randomized to remove correlation of plaintext with the cipher power consumption [10]. The objective of embedded security is to minimize, both temporally and spatially, the risk of an attack. This objective crosses abstraction layers and individual validation layers.

The outline of the remainder of the paper is as follows. In Section II, 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 secrets upon which the total security is based 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 secrets. In Section III, we present a methodology for secure system design, based on multi-level system validation. The protection can be partitioned into individual design problems at different levels of abstraction (protocol, architecture, micro-architecture, 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 between secure and non secure parts without compromising the secrets. In Section IV, 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 V presents the main results obtained out of the ThumbPod-2 design.

II. THUMBPOD-2 EMBEDDED AUTHENTICATION

In this section, we briefly introduce the driver application for this 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 pre-stored 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.

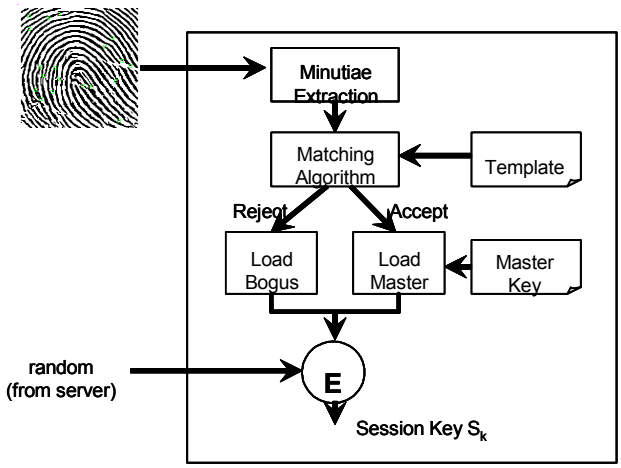

Figure 1. Overview of the security protocol for the Thumbpod-2 client.

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. Figure 1 gives an overview of the matching protocol. ThumbPod is a client in a client-server application. In normal operation, the user puts his finger on the sensor and the device will capture his or her fingerprint. The device will extract the minutiae out of this fingerprint and compare it with the pre-stored template. Depending on the resulting matching score, ThumbPod will give the user access to a securely stored master key. 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 is then used in the communication between the server and the ThumbPod.

From Figure 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 in the ThumbPod.

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 secure handling of the secrets 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

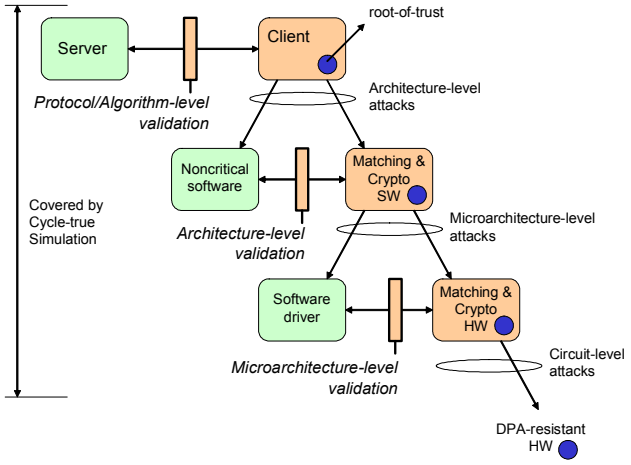

Figure 2. Multilevel validation of secure embedded systems.

would not expect it (cfr 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 multi-level approach is discussed in the next section.

III. MULTI-LEVEL DESIGN VALIDATION

In multi-level design validation, we consider the operation of an embedded system at multiple abstraction levels, and aim to validate 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. Security attacks beyond that single level are considered to be side-channel attacks, that must be dealt with at a different level of design abstraction. This idea is expressed in Figure 2. At the protocol level, the entire ThumbPod-2 is considered trusted, because it holds a secret or a root of trust. The validation of the communication protocol thus aims at showing that this root-of-trust cannot leak by illegal protocol sequences. The ThumbPod-2 device itself can still be attacked at the architecture level, by interfering, for example, with the embedded software present on the device. These attacks are thwarted by partitioning the device at the architecture level in a trusted and a non-trusted part, which will isolate the root-of-trust to a confined area. 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 the security assumptions that were made for the higher levels also at the lower levels.

In the following paragraphs, we give examples of each of the validations 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.

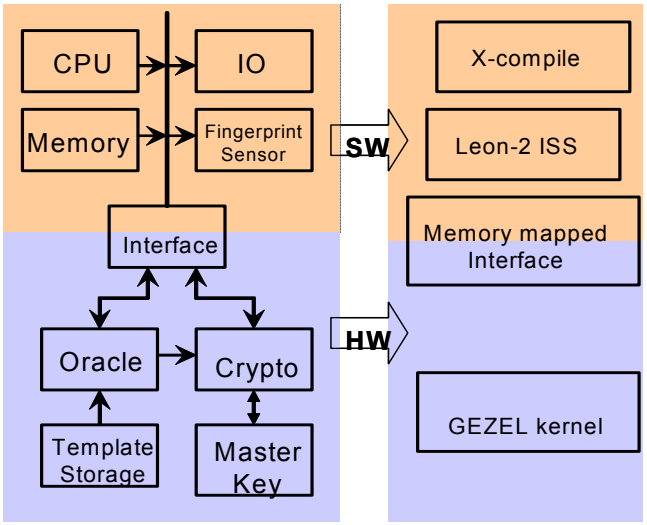

Figure 3. ThumbPod-2 platform (left) and GEZEL-based system simulation (right).

A. Cycle-true Platform Validation in GEZEL

The target architecture for ThumbPod-2 consists of an embedded processor core, to implement the software parts, combined with a secure hardware accelerator. This hardware accelerator has been developed using special circuit-level techniques that protect the accelerator against circuit-level side-channel attacks [9]. In this paper, we will focus on the validation techniques needed at the protocol/algorithm, architecture and micro-architecture level, and assume that the circuit-level will be side-channel-attack free by construction.

Under these assumptions, our validation environment corresponds to a cosimulation environment: we need the ability to design and cosimulate hardware and software components. We constructed a cycle-true model of the entire platform. Figure 3 shows the ThumbPod-2 platform including the 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 simulation environment for this system is shown on the right of Figure 3, and consists of an instruction-set simulator for the embedded core, combined with a simulation kernel for the hardware design, called GEZEL [11][12]. 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. The system simulation is cycle-accurate. The cosimulation speed of the complete platform using GEZEL and the LEON2 ISS is about 50 KHz on a 3 GHz Pentium PC when the system is fully exercised.

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

Figure 4. Original biometric protocol.

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

B. Protocol Transformation and the Root-of-Trust

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 secure side.

Consider the biometric verification protocol as shown in Figure 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 non-sensitive due to the non-invertible 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.

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 onto a secure confined part portion of the device, with well defined interfaces, i.e. a hardware root-of-trust. At this point, we describe how the protocol of Figure 4 is refined into a refined protocol shown in Figure 5. Indeed on Figure 4, for a secure correct operating device, it is assumed that the whole device must be secure.

In the first step of the refinement, the designer must make a security partitioning and determine which data elements must be contained within the root-of-trust and which can be left exposed. This is a security analysis validation process, where the designer at the security level determines which variables are sensitive to compromise. In our system, the master key K and the template must remain in the root-of-trust. In addition, all functions dealing directly with the root-of-trust variables must also be mapped to the root of trust; these functions are the matching algorithm, the keyed-hash generation, the session key generation, and token generation. These are shown in bold on Figure 4. On Figure 5, this results in the split of the unit Device into a secure portion of the device, Device(S), and an insecure portion of the device, Device(I). Device(S) handles sensitive data, while Device(I) deals with non-secure data. For instance, Device(I) obtains the raw image of the input fingerprint and extracts the minutiae. These minutiae are then passed from the insecure to the secure part of the device. Also, random numbers and tokens are exchanged between the secure and non-secure parts. In our design, Device(S) is a custom-designed secure hardware coprocessor while Device(I) is an insecure embedded LEON processor.

Once these variables are mapped into the root-of-trust, a set of multi-level secure interfaces and protocols must be constructed between the root-of-trust's secure functions and the embedded processor's insecure functions. For the

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 k
	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

Figure 5. Transformed biometric protocol: Server, Device(I) and Device(S).

matching algorithm, a secure oracle process was designed (as described in the next section). For the generation of the session key and ensuing token, the interface consists of the processor passing insecure data (RAND | RANDT) and instructions, and receiving an encrypted token, TOKEN1. After TOKEN1 is received the insecure processor sends another instruction to the root-of-trust indicating a request to perform the keyed-hash. The processor receives this as TOKEN2. The interface must be constructed in hardware such that the insecure processor is unaware if TOKEN1 and TOKEN2 are real tokens or merely dummy tokens meant to indicate a fraudulent transaction. The final enhanced biometric protocol is shown in Figure 5.

In terms of microarchitecture, the secrets and manipulations of them are confined to the memory-mapped coprocessor. We interface the hardware coprocessor to the 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 construction 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 at the architecture-level takes places on these three buses. Hence, the communication across these buses and their ensuing security are what must be validated.

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

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 `$begin_encrypt` instruction allows the secure coprocessor to generate the first token (or a dummy token). After the token is generated and therefore consumed by the processor, the processor sends a `$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 is indicated in Figure 5.

C. Biometric Matching Oracle

The biometric matching process compares the minutiae set of a user to a template minutiae set. This template is pre-

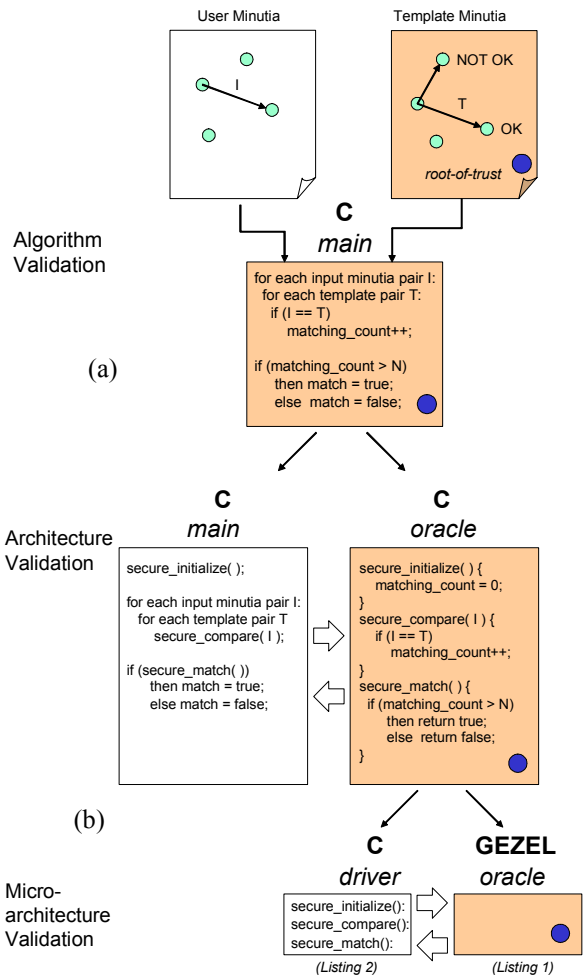

Figure 6. Multilevel design validation of the biometric oracle. (a) Complete algorithm needs protection. (b) After security partitioning.

stored and belongs to the true owner of the ThumbPod. The original matching algorithm of the ThumbPod is based on a 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 [13]. 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 [13]). The C code is also executed 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 minutiae (the root-of-trust). Subsequent refinements of the algorithm therefore aim to isolate 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 con-

cept of an *oracle*. The oracle will hide the matching 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 minutiae 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 micro-architecture 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 the next section, we turn to the issue of micro-architecture modeling in GEZEL, and cosimulation of the coprocessor with C.

IV. COPROCESSOR MODELING, IMPLEMENTATION, AND VERIFICATION IN GEZEL

In the previous two sections, we described a partitioning process that isolates the root-of-trust over multiple abstraction levels. At the micro-architecture level, this partitioning results into 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 (a) automatic generation of synthesizable VHDL and (b) generation of RT-level and chip-level test-vector stimuli.

A. Micro-architecture 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 Figure 6, this step tests the value of variable `match_count` against a preset value `N`. The variable `match_count` holds the number of minutiae in the input fingerprint that are considered matched (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. For example, an attacker could

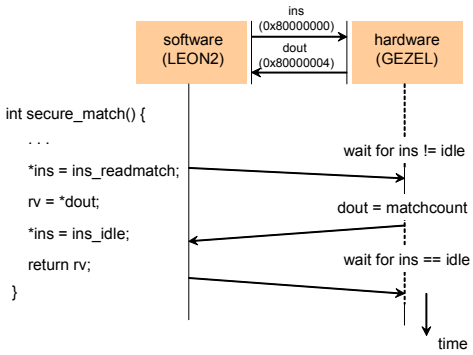

Figure 7. One-way handshake between C and GEZEL.

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 micro-architecture level, the function `secure_match()` is separated into a secure hardware part described in GEZEL, and an insecure software driver in C.

Through the memory-mapped interfaces, communication between HW and SW is modeled. For instance, a one-way handshake between SW and HW is illustrated in Figure 7. In this example, the one-way handshake protocol assumes that the hardware is required to synchronize the software but not vice versa. Initialized C-pointers, in this example for "ins" and "dout", indicate the memory locations that are shared between the HW and the SW.

B. 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 to record 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.

Using a trace directive in I/O signals, interface signals are probed and placed into a text file with one value written per cycle. These stimuli can be reused for RTL-simulation of the VHDL, and later for chip-level validation of the actual chip. For RTL-simulation at VHDL, such stimuli have the advantage that they can be exercised at the block-level. An RTL processor model of an embedded RISC such as the LEON 2 runs at a few 10 Hz at most on a VHDL simulator. Considering that our testbenches range between 100K to 100M cycles, RT simulation at VHDL of the complete system is clearly out of the question. However, we did perform simulations at the individual block level, primarily to verify the correctness of the generated VHDL code.

V. RESULTS

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

A. 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 micro-architecture 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.

B. Implementation and Design Validation Flow

The implementation flow for the ThumbPod-2 system is shown in Figure 8. After platform cosimulation using C and GEZEL, the GEZEL modules are converted into synthesizable RTL-VHDL, and testbench stimuli are recorded into files. These stimuli are used for block-level RTL simulations in VHDL. LEON's output to the coprocessor (instruction and input data buses) were 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 LEON processor was not implemented on the same 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. In this way, chip testing could use the same `coproc_driver.c` test-bench 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.

C. Coding and Design Complexity

Finally we document the coding and design complexity of the design in Table II and Table III. Table II 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 III shows the size of the design descriptions and the implementation for each of those components.

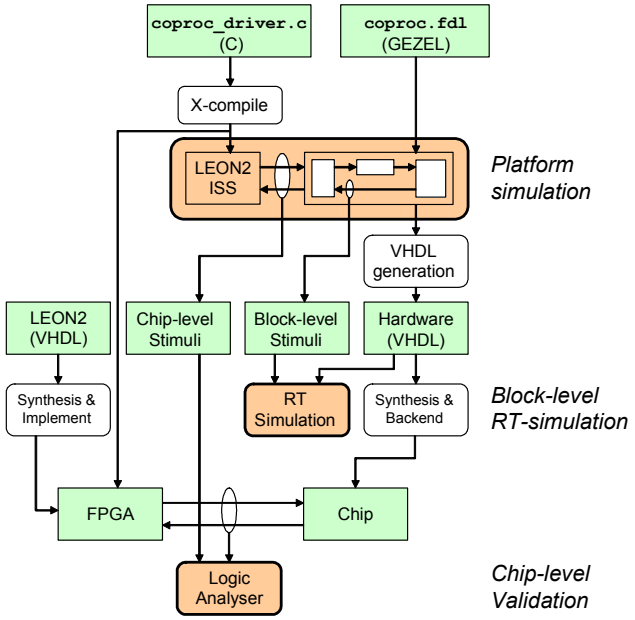

Figure 8. Design flow with back-end validation levels.

TABLE II. 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 III. 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

VI. CONCLUSIONS

System-level design of secure embedded systems 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 has demonstrated the ability to

withstand a wide range of security attacks.

VII. ACKNOWLEDGEMENTS

The authors wish to acknowledge the NSF, SRC, UC Micro, the Fannie and John Hertz Foundation (DH), and the anonymous reviewers of this text.

REFERENCES

[1] S. Ravi, e. al., “Challenges in Designing Secure Embedded Systems," Invited paper, ACM Transactions on Embedded Computing Systems (Special Issue on Security), August 2004.

[2] Arnold, T.W. and Van Doorn, L. P., The IBM PCIXCC: A new cryptographic coprocessor for the IBM eServer, IBM Journal of Research and Development, Vol 48, No 3/4, 2004, pp. 491-503.

[3] A. Huang. “Keeping Secrets in Hardware: the Microsoft XBox Case Study.” AI Memo 2002-008, 2002, MIT.

[4] S. Edwards, L. Lavagno, E. Lee, A. Sangiovanni-Vincentelli, “Design of Embedded Systems: Formal Models, Validation, and Synthesis,” Proc. IEEE, 85(3):366—390.

[5] P. Mishra, N. Dutt, N. Krishnamurthy. M. Abadir, “A Top-Down Methodology for Microprocessor Validation,” IEEE Design and Test of Computers, March-April 2004.

[6] P. Schaumont, I. Verbauwhede, “Domain specific codesign for embedded security,” IEEE Computer, 36(4):68—74, April 2003.

[7] S. Pearson, “Trusted Computing Platforms, the Next Security Solution,” HP Tech. 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 International High Level Design Validation and Test Workshop (HLDVT 2003), pp. 77-82, November 2003.

[9] K. Tiri, D. Hwang, A. Hodjat, B.C. Lai, S. Yang, P. Schaumont, I. Verbauwhede, “A Side-Channel Leakage Free Coprocessor IC in .18um CMOS for Embedded AES-Based Cryptographic and Biometric Processing, Proc. DAC 2005.

[10] S. Coron, L. Goubin, “On boolean and arithmetic masking against differential power analysis,” Cryptographic Hardware and Embedded Systems - CHES 2000, LNCS 1965, 231—237, 2000.

[11] The GEZEL Design Environment, Online. <http://www.ee.ucla.edu/~schaum/gezel>.

[12] P. Schaumont and I. Verbauwhede, “Interactive cosimulation with partial evaluation,” Proc. Design Automation and Test in Europe (DATE 2004), pp. 642-647, February 2004.

[13] S. Yang and I. Verbauwhede, "A secure fingerprint matching technique," Proc. ACM Workshop on Biometrics: Methods and Applications, pp. 89-94, November 2003.