Open access • Journal Article • DOI:10.1109/MC.2003.1193231

Domain-specific codesign for embedded security — Source link

Patrick Schaumont, Ingrid Verbauwhede

Institutions: University of California, Los Angeles

Published on: 01 Apr 2003 - IEEE Computer (IEEE)

Topics: Secure cryptoprocessor, Application software, Virtual machine, Interface (Java) and Software prototyping

Domain-Specific Codesign for Embedded Security

Effective security begins with system design and involves many layers of abstraction. A complete system is a codesign of domains requiring domain-specific processors and a flexible platform architecture to support them.

Patrick Schaumont
Ingrid Verbauwhede
University of California, Los Angeles

Security is the Internet's ugly duckling—few Internet users think about it unless they've experienced a virus or a hacker. Whereas we tend to notice the lack of telephone service immediately, we usually discover the lack of security when it is too late.

Living in a networked world worsens this situation, as the Slammer worm demonstrated in January 2003. In less than 24 hours, the worm infected 250,000 servers, bringing Internet traffic to a near-complete stop.[1] The fact that most new networked applications are implemented in embedded and portable contexts further aggravates this problem. Wireless local area network access points, personal digital assistants (PDAs), and sensor nodes may become the backdoors of tomorrow's Internet, compromising secure operation.

SYSTEM DESIGN

Effective security support is a system design problem that needs an integrated approach. Securing a system requires more than simply adding encryption processors and virus-scanning software. Rather, you must implement those security elements in an organized way.

A system's security is only as strong as its weakest link. For example, a smart card's strongest cipher algorithm is worthless if a hacker can disassemble the card and retrieve sensitive data by observing its power consumption.[2] We think of security as a design domain with multiple layers of design abstraction, and a complete system as a codesign of domains (security, networking, and graphics, for example) rather than a codesign of implementations (such as hardware and software).

The classic view emphasizes using hardware for performance and software for flexibility. In embedded applications such as mobile phones, PDAs, and sensor network nodes, however, energy efficiency is of tantamount importance.

We approach these challenges by designing domain-specific processors and integrating them into a flexible platform architecture based on a reconfigurable interconnect. Domain-specific architectures consume energy much more efficiently than general-purpose architectures. The processors have restricted and specialized instruction sets that cover a class of security support algorithms, such as Advanced Encryption Standard (AES) encryption and decryption[3] with cipher block chaining message authentication code (CBC MAC), output feedback (OFB), and counter (CTR) modes. Figure 1 shows an example of the energy efficiency of AES encryption on five different platforms.

SECURITY PYRAMID

The pyramid in Figure 2 represents the security design domain using the 3G mobile phone communications system as an example. This system contains various types of security, such as user authentication, confidentiality, and location privacy. The 3G Partnership Project details these security requirements and outlines the responsibilities of all parties—subscriber identity module card, handheld, base station, and location registers—involved in the communications link.[4]

Published by the IEEE Computer Society
0018-9162/03/$17.00 © 2003 IEEE

The pyramid's top level considers an end-to-end model of the main parties in the communications link. This level expresses a protocol as a coarse-grained control flow that sequences cryptographic algorithms. This protocol specifies how the setup phase will proceed and what encryption schemes will be used. It relies on establishing mutual trust, which the system implements by exchanging certificates and generating and verifying keys. Different applications require different levels of trust.

The *algorithm* level specifies the protocol level's building blocks. This second level contains ciphers and one-way hash functions. For key agreement, the algorithm level uses random-number generators and modular arithmetic functions. Symmetric ciphers can be either stream- or block-based, processing *n*-bit blocks of data for each invocation. In addition, defining a mode of operation can enhance the cipher with feedback and an initialization procedure.[5] 3G wireless proposes using Rijndael encryption (www.esat.kuleuven.ac.be/~rijmen/rijndael) on the mobile phone SIM card for key agreement and requires Kasumi encryption inside the handheld mobile phone for real-time encryption.[4]

As Figure 3 shows, key agreement and ciphering belong to different planes of operation. Whereas key agreement protocols are rarely real time, communications ciphers have real-time constraints. Therefore, the processing requirements for SIM card cryptography are much lower than for handheld cryptography.

In a shared private-key scheme, the SIM card stores one copy of the secret key. A public-key scheme does not require such mutual secrets. However, a private-key scheme is considerably more energy efficient than a public-key scheme. For example, a Diffie-Hellman-based elliptic curve public-key agreement is three orders of magnitude more computation and energy intensive than a Rijndael-based Kerberos key-agreement protocol.[6] Protocol-level modeling offers the context required for making such algorithmic tradeoffs.[7]

The *architecture* level considers the mapping of the algorithms and security protocol. This mapping is a layer of software with appropriate hardware specialization. The security application pyramid expresses the protocol and algorithm levels as a Java application on top of Java Cryptographic Extensions.[8] JCE offers cryptographic services and as such represents a specialized software library. The JCE *cryptographic service provider* implements part of the JCE standard API and can have both software and hardware implementations (www.bouncycastle.org). We use standard Java code during key agreement and setup phases, when performance and power consumption is noncritical. When the connection is established and the link acquires a real-time and longer-lasting character, we use an encryption processor, configured as a JCE hardware provider.

The *microarchitecture* level expresses a more detailed, hardware-oriented view. We extend traditional microarchitectures with energy-efficient

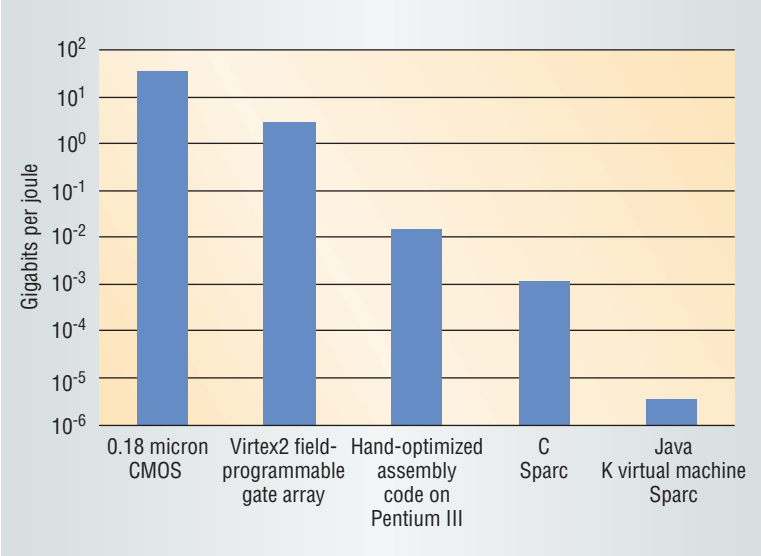

Figure 1. Energy efficiency of the Advanced Encryption Standard on five platforms. The energy efficiency, which spans seven orders of magnitude, is expressed as the number of gigabits the system can encrypt per joule (www.ee.ucla.edu/~schaum/rijndael).

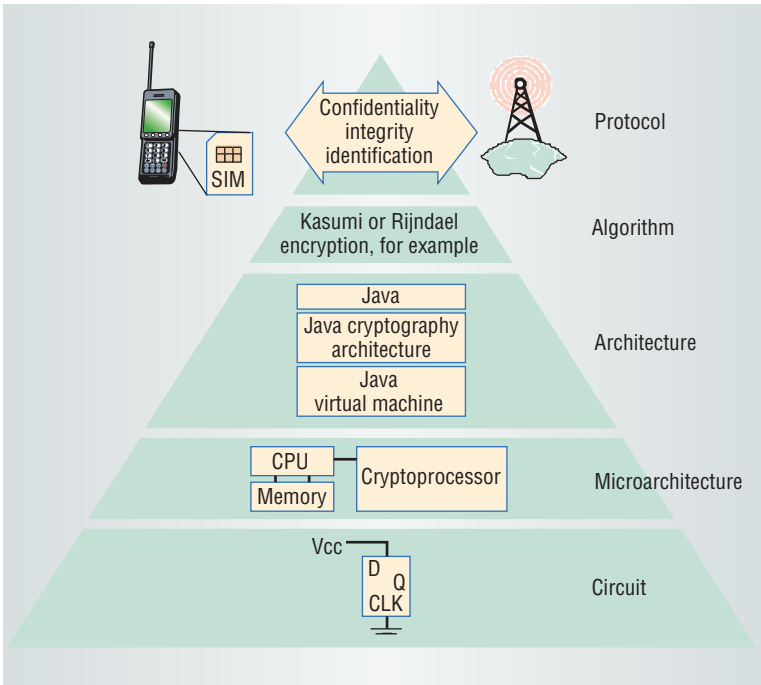

Figure 2. The security application domain pyramid. Each abstraction layer represents specific modeling, design, and implementation issues that must be covered for secure system operation.

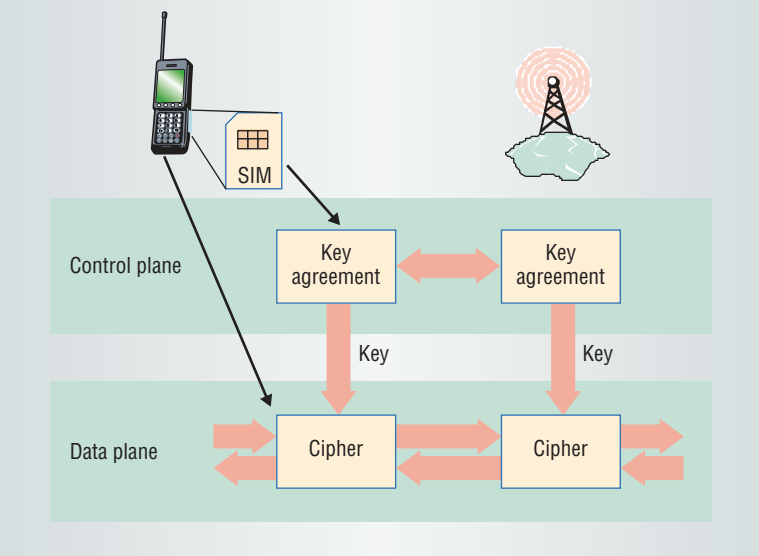

Figure 3. Key agreement and ciphering in mobile phones. The two protocols operate on different planes and require different levels of processing.

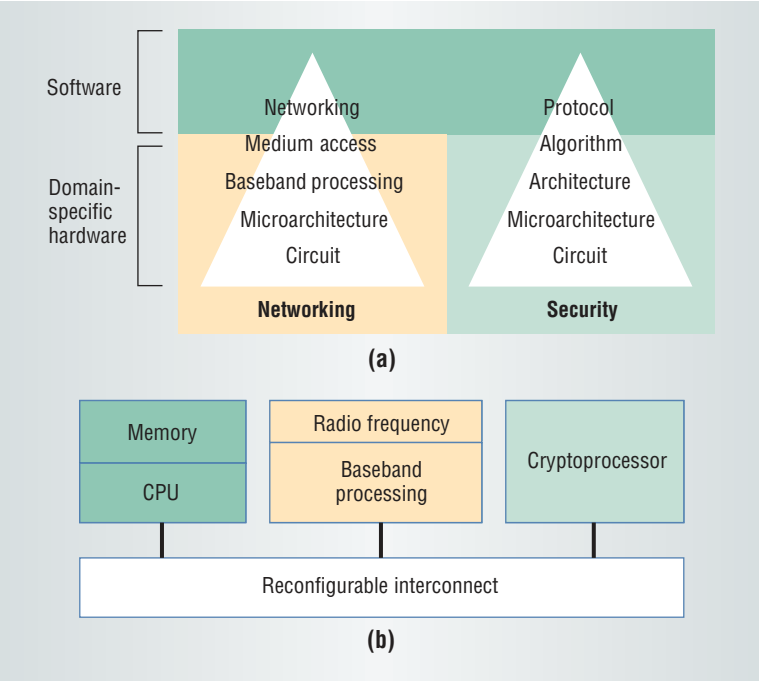

Figure 4. 3G handheld as a Rings system. (a) The application view includes networking and security domains; (b) the architecture view shows the hardware for baseband processing and cryptography.

cryptoprocessors to support cryptography functions. The accelerator processor's flexibility is an important consideration. Contemporary standards require support for multiple cryptoalgorithms; however, energy efficiency and flexibility are inversely related. We therefore use domain-specific coprocessors with a restricted and highly specialized instruction set.

Finally, the *circuit* level addresses implementation and architectural integration of the cryptoprocessors. At this level, the cryptocircuit needs to resist power and timing attacks. These techniques do not have to affect the full cell phone design, but they are useful where a high amount of trust is required, such as in the SIM cards.

NEXT-GENERATION EMBEDDED SYSTEMS

In addition to the cryptographic domain, next-generation embedded systems will have to deal with other design domains, such as networking, wireless baseband, signal processing, and multimedia. Domain specialists can devise specialized processors for many design domains. We have developed instruction-set architectures and microarchitectures for the security domain. We have also developed a system architecture to support these processors.

Reconfigurable interconnect

Embedded systems consisting of multiple design domains often contain multiple specialized processors. Figure 4a depicts the networking and security domains of an embedded networked device such as a 3G handheld phone.

We can use a pyramid view to express each domain at different levels of abstraction. System design uses software and specialized domain-specific hardware to combine and implement these pyramids.

The system architecture shown in Figure 4b includes a general-purpose processor and dedicated hardware for baseband processing and cryptography. A reconfigurable interconnect combines the two blocks.

Our Reconfigurable Interconnect for Next-Generation Systems (Rings) architecture consistently distributes data processing over several specialized units to facilitate heterogeneous processing.[9] Communication relies on flexible point-to-point links. Instead of a global clock, the architecture uses loosely coupled synchronous islands. The system does not include a tightly coupled central controller. The CPU keeps the system properly configured and organized, but maintains only loosely controlled coupling with the individual domain processors.

We can formulate the hardware/software codesign problem as finding a mapping from the pyramids in Figure 4a to elements in Figure 4b. Domain-specific hardware guides this mapping. An encryption processor covers the security pyramid's lower abstraction levels. Using the encryption processor according to a specific security protocol requires further software integration. This is also true for a networking pyramid: Dedicated hardware enables front-end and baseband processing. At a higher abstraction level, communication pro-

tocols running in software use these components. Thus, in a complete application, software integrates all domain pyramids at the top level.

The reconfigurable interconnect network establishes arbitrary point-to-point links, supporting both system architecture flexibility and distributed system operation. To the domain-specific processors, this reconfigurable interconnect looks like a service layer in the same way that an operating system provides services for software tasks. Distributed systems operate using multiple, concurrent control and data streams between system components. This way, the interconnect configuration can reflect various computational models.[10]

Dataflow and process networks require point-to-point links. Other computational models, such as publish-subscribe, require broadcasting connections. We implement the network using on-chip code division multiple access technology.[11] The CDMA links are scalable in both bandwidth and connectivity. Although the links implement the equivalent of an on-chip routing packet network, link modulation reduces the routing overhead associated with a packet-switched on-chip network.

Cryptoprocessor microarchitecture

The Rings architecture integrates domain-specific processors. Each processor has one or more command and data ports that connect the internal microarchitectures to Rings. Figure 5 shows the typical microarchitecture of such a cryptoprocessor.

Cryptoprocessors use modularity and a control hierarchy in their implementation. Because the cryptoprocessor operates in a loosely coupled environment using a limited set of powerful instructions, the system must move more intelligence into local controllers. Inside the cryptoprocessor, a hierarchy of controllers separates I/O communication from cryptographic operations and helps cope with cryptographic algorithm control complexity, which arises from the iterative nature of cryptoalgorithms, the complex number systems some cryptoalgorithms use,[12] and the need to support multiple algorithm variants in a single implementation. The architecture template in Figure 5 applies to both private-key and public-key processors.

Efficiently operating the cryptoprocessor requires interleaving I/O processing with the encryption algorithm operation. This produces a block-pipelined system: While the input processor reads in block N, the cryptocore works on block $N - 1$, and the output processor works on block $N - 2$. Two issues affect the application of block-pipelining to cryptoprocessors:

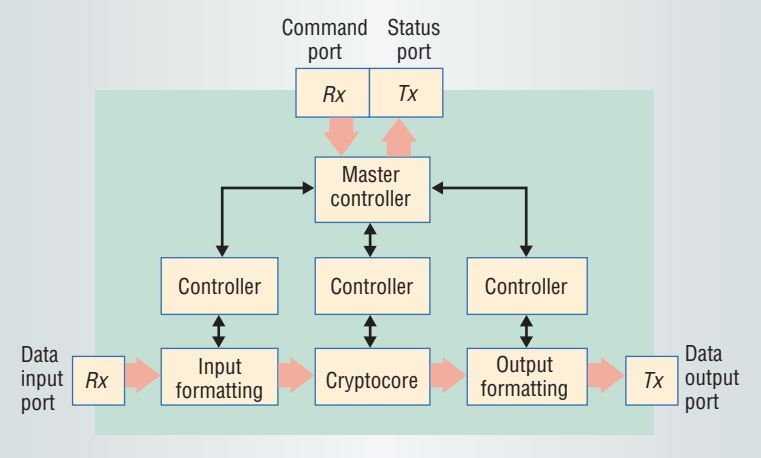

Figure 5. Microarchitecture of a Rings cryptoprocessor. A hierarchy of controllers separates I/O communication from cryptographic operations and helps make cryptographic algorithm control complexity more manageable.

- Unlike a signal-processing algorithm, in which the system often can ignore such transients, each data block in the cryptoprocessor has equal importance (secrecy). Thus, the processing pipeline must be carefully controlled.
- Dynamic stalls can occur in the pipeline if one of the pipeline stages contains a handshaking I/O operation.

Both issues affect the cryptoprocessor's programming model. The classic hardware function call model no longer holds. Rather, a programming sequence must take the form of a small macroprogram.

Cryptoprocessor instruction set architecture and programming

One requirement for achieving system-level distributed operation is minimizing the bandwidth that global control and status streams require. Thus, instructions flowing from a system controller to a domain-specific processor should be semantically rich.

A cryptographic processor's basic operation is always the same: It reads in a block of data, performs a crypto-operation, and returns the processed result. Therefore, we can think of the crypto-operation as an operation mode, or *continuous instruction*, for the processor. Unlike single instructions, which specify a unique event such as "the next data packet presented at the input is an encryption key," a continuous instruction specifies a behavior over a sustained period—for example, "start ECB-mode AES encryption."

Continuous instruction efficiently creates distributed systems that are driven by their data streams rather than a central controller's control pulse. We can model support for the pipeline startup and termination effects as continuous instructions: one instruction to start the block

Table 1. Typical cryptoprocessor instruction set.

Instruction	Type	Description
Reset	Single	Initialize the processor
LoadKey	Single	Store next data packet as current encryption key
GenSubKey	Single	Create roundkey material out of current key
Encrypt	Configuration	Next cipher operation is encryption
Decrypt	Configuration	Next cipher operation is decryption
ECB	Configuration	Next cipher operation uses electronic code book (ECB) mode
CBCMAC	Configuration	Next cipher operation uses cipher block chaining message authentication code (CBC MAC) mode
IVReg	Configuration	Next data packet contains initialization vector (for CBC MAC)
Start	Continuous	Start cipher pipeline
Finish	Continuous	Flush cipher pipeline

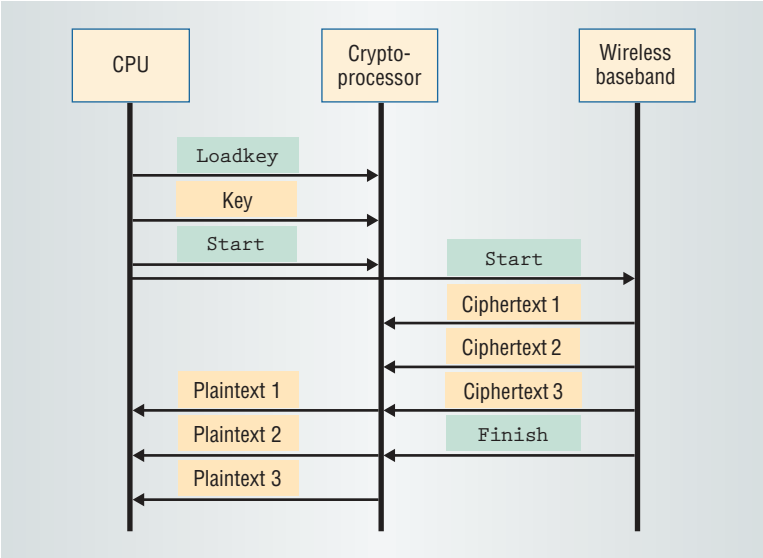

Figure 6. Distributed operation in Rings shown as a message sequence chart. The CPU initiates streamlined operation between the wireless baseband processor and the cryptoprocessor by sending data and control packets.

pipeline and the other to terminate it.

Table 1 lists a subset of the instruction set of an actual encryption-decryption chip. The table identifies three types of instructions: single, configuration, and continuous. By default, the processor sits idle, waiting for an instruction. Depending on the instruction type, the processor accepts one or more data packets at the processor data input port and generates packets on the data output port. Instructions such as Reset and GenSubKey take no operands and affect only the cryptoprocessor's internal state. An instruction such as LoadKey requires the next data packet to be an encryption key. The encryption processor strictly enforces the length and format of these data packets and thus prevents partial key load, which some cipher attacks use.

Configuration instructions select an algorithm (such as encrypt or decrypt) or enable a mode of operation (electronic code book or CBC MAC, for example). The continuous instruction Start initiates the cryptopipeline. Two input data packets are required before the first encryption result will be available. The continuous instruction Finish flushes the cryptopipeline, allowing final output data packets to leave the cryptopipeline without feeding in new input data.

With this instruction set, we can write small macroprograms, such as the following command sequence used to encrypt a data stream in ECB mode:

```
Reset;
LoadKey; // send key
GenSubKeys;
ECB;
Encrypt;
Start; // send data packets
Finish;
```

The CPU sends this command sequence to the cryptoprocessor (Figure 4b). The LoadKey and Start commands also need the CPU to send data packets. A key feature of this programming model is that it does not specify where the data packets originate. Thus, while the data packet with the key might come from the CPU, the actual data stream to encrypt might come from the wireless baseband processor. This leads to programming sequences, illustrated in Figure 6.

CODESIGNING APPLICATION DOMAINS

Systems with multiple design domains require codesign of application domains. Dedicated hardware processors implement the application domains, and software integrates them.

Figure 7 shows the architecture of our prototype embedded security application, the ThumbPod, which we use for remote identification applications such as intelligent keys or electronic payments (www.ivgroup.ee.ucla.edu/thumbpod). The device combines security, biometrics, and networking domains. ThumbPod's architecture supports security, combining the Java cryptography architecture (JCA), built on top of an embedded virtual machine, with an AES cryptoprocessor. The top-level security protocol layer is written in Java.

Additional software support consists of a dynamic application download using the Java application manager. Sun's K virtual machine (KVM) offers an infrastructure for secure code download and execution and integrates the cryptoprocessor through a native interface. We program the cryptoprocessor with a macrolanguage (see Table 1).

This model presents several codesign challenges:

- Dynamic software needs integrity verification before it can execute on a secure platform. In Java applications, JCA performs bytecode verification. ThumbPod's software is heterogeneous and contains bytecodes as well as macroprograms for the domain-specific processors. Both can have their own provider and, consequently, their own certificates and integrity-checking processes.
- The hardware processors use block-pipelining techniques, increasing parallelism in the system but requiring more complex synchronization. In particular, the service provider interfaces in KVM that integrate the cryptoprocessor must account for such parallelism.
- A cryptoprocessor uses sensitive key material that must not leave the processor. If this key material needs to travel over the reconfigurable interconnect, it creates a potential security hole. This security hole is unlike traditional software security holes because it travels across many levels of abstraction.
- Writing software for ThumbPod is equivalent to writing different kinds of configuration content. The Leon-2 runs C programs, while the cryptoprocessor and fingerprint sensor run domain-specific instructions. These features need to be unified under a single Java programming model without compromising distributed operation and performance.

These codesign challenges share an important property. Each requires an understanding of the system operation at multiple levels of abstraction—particularly of the security pyramid. Neither abstract mathematical models nor detailed cycle-true models alone can handle these challenges. For the former, modeling will fall short; in the latter, system complexity will become a showstopper. Consequently, these codesign problems are more likely to be solved by a team of software, hardware, and application domain specialists than by an individual.

Future embedded systems will have design characteristics similar to ThumbPod, with multiple application domains converging in a single implementation. Domain-specific hardware processors will be a key enabler of energy efficiency. Currently, we are developing a codesign environment to support development and verification of these coprocessors at a high level. We are also formulating the interconnect services of the Rings architecture as a high-level computational model. ■

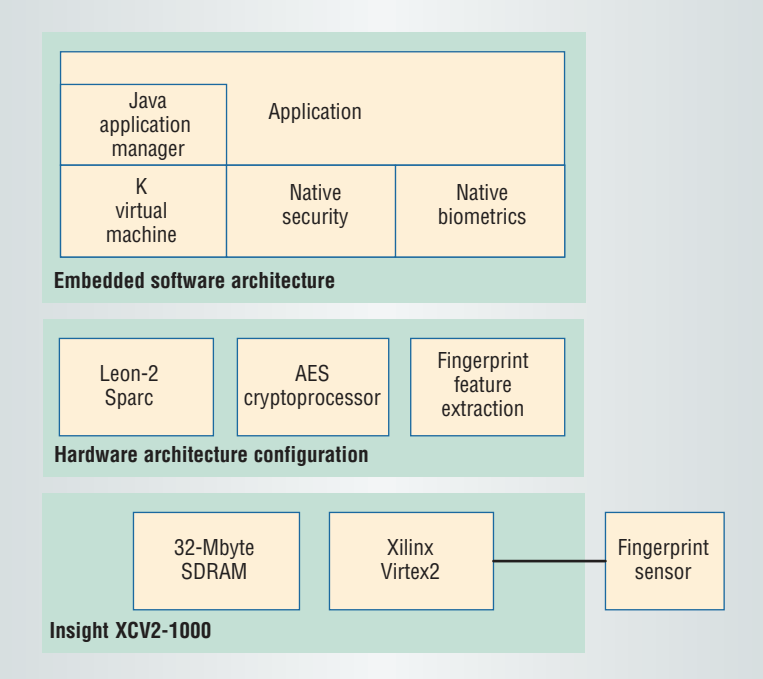

Figure 7. The ThumbPod prototype combines security, biometrics, and networking domains. A Java cryptography architecture and an AES cryptoprocessor support device security.

Acknowledgments

The US National Science Foundation (grant #0098361) and a 2002 Design Automation Conference Graduate Research Fellowship supported this work. Gaisler Research provided the Leon-2 Sparc and its TSIM instruction-set simulator.

References

1. "Virus Hits ATMs and Computers across the Globe," *The New York Times*, business section, 26 Jan. 2003; www.nytimes.com/2003/01/26/business/FT1042491212045.html.
2. R. Anderson and M. Kuhn, "Tamper Resistance—A Cautionary Note," *Proc. 2nd Usenix Workshop Electronic Commerce*, Usenix, 1996, pp. 1-11.
3. *Advanced Encryption Standard*, National Inst. of Technology and Standards, Federal Information Processing Standards (FIPS) Pub. 197, 2001; www.nist.gov/aes.
4. *3G Security: Security Architecture*, Universal Mobile Telecomm. System (UMTS), tech. specification 3GPP TS 33.102, 1999.
5. *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*, National Inst. of Technology and Standards, SP 800-38A, Dec. 2001; www.nist.gov/modes.
6. A. Hodjat and I. Verbauwhede, "The Energy Cost of

Secrets in Ad-Hoc Networks," *Proc. IEEE CAS Workshop Wireless Comm. and Networking*, IEEE Press, 2002; http://dsp.jpl.nasa.gov/cas/Papers.html.
7. B. Kienhuis et al., "A Methodology to Design Programmable Embedded Systems," *SAMOS: Systems, Architectures, Modeling, and Simulation,* LNCS 2268, Spinger, 2001, pp. 18-37.
8. Sun Microsystems, "Java Cryptography Extension," 2001; java.sun.com/products/jce/index-14.html.
9. I. Verbauwhede and M.F. Chang, "Reconfigurable Interconnect for Next-Generation Systems, *Proc. ACM/Sigda Int'l Workshop System-Level Interconnect Prediction* (SLIP), ACM Press, 2002, pp. 71-74.
10. E.A. Lee, *Embedded Software, Advances in Computers*, M. Zelkowitz, ed., vol. 56, Academic Press, 2002.
11. M.F. Chang et al., "RF/Wireless Interconnect for Inter- and Intra-Chip Communications," *Proc. IEEE*, IEEE Press, vol. 89, no. 4, 2001, pp. 456-466.
12. S. Janssens et al., "Hardware/Software Co-Design of an Elliptic Curve Public-Key Cryptosystem," *Proc. Workshop Signal Processing Systems*, IEEE CS Press, 2001, pp. 209-216.

Patrick Schaumont *is a doctoral candidate in the Electrical Engineering Department of the University of California, Los Angeles. His main research interests are design methods and tools for embedded and domain-specific computing systems. He received an MS in computer science from Rijksuniversiteit Ghent, Belgium. Schaumont is a student member of the IEEE. Contact him at schaum@ee.ucla.edu.*

Ingrid Verbauwhede *is an associate professor in the Electrical Engineering Department at UCLA. Her main interests are in architecture design, VLSI implementation, and design methods for low-power embedded systems. She received a PhD from K.U. Leuven, Belgium. Verbauwhede is a senior member of the IEEE. Contact her at ingrid@ee.ucla.edu.*