Implementing a Protected Zone in a Reconfigurable Processor for Isolated Execution of Cryptographic Algorithms

A. Onur Durahim, Erkay Savaş Sabanci University
Orhanli, Tuzla Istanbul, 34956 Turkey
durahim@su., erkays@sabanciuniv.edu

Kazim Yumbul Gebze Institute of Technology
Gebze, Kocaeli, 41400 Turkey
kyumbul@gyte.edu.tr

***Abstract*—We design and realize a protected zone inside a reconfigurable and extensible embedded RISC processor for isolated execution of cryptographic algorithms. The protected zone is a collection of processor subsystems such as functional units optimized for high-speed execution of integer operations, a small amount of local memory, and general- and special-purpose registers. We outline the principles for secure software implementation of cryptographic algorithms in a processor equipped with the protected zone. We also demonstrate the efficiency and effectiveness of the protected zone by implementing major cryptographic algorithms, namely RSA, elliptic curve cryptography, and AES in the protected zone. In terms of time efficiency, software implementations of these three cryptographic algorithms outperform equivalent software implementations on similar processors reported in the literature. The protected zone is designed in such a modular fashion that it can easily be integrated into any RISC processor; its area overhead is considerably moderate in the sense that it can be used in vast majority of embedded processors. The protected zone can also provide the necessary support to implement TPM functionality within the boundary of a processor.**

***Keywords*-cryptography; cryptographic unit; isolated execution; secure computing; trusted computing;**

I. Introduction

Secure and efficient implementations of cryptographic algorithms become more of a focal point for research in cryptographic engineering since various attacks [14], [4], [1] successfully compromise *realizations* of many cryptosystems which are theoretically proved secure otherwise. Since general-purpose processors fulfill neither the timing nor the security constraints of cryptographic applications due to different set of design considerations, special-purpose cryptographic co-processors are built to remedy this problematic. Nevertheless, the cryptographic processors turn out to be not entirely free from the security concerns and furthermore, introduce their own problems such as security risks and speed considerations accrued in host processor/co-processor setting.

Aware of the inadequacy of the software-only solutions, computer manufacturers already introduced hardware extensions to their processor cores to accelerate cryptographic computations and to provide secure execution environment. A notable development is that new architectures introduced by three major manufacturers [6], [8], [2] allow that security-sensitive applications execute in an environment strictly free from the intervention of other simultaneously running processes. This feature is known as process isolation and enforced by the hardware. A strictly enforced process isolation is definitely beneficial in thwarting an important class of attacks known as micro-architectural side-channel attacks [7], [1]. From these observations, developments and results, the need for further research in new computer architectures that support efficient and secure implementations of cryptographic algorithms becomes obvious.

In this paper, we investigate the realization of a protected zone in a reconfigurable embedded processor that provides cryptographic algorithms with highly secure execution environment. The protected zone consists of architectural subsystems of a local memory, registers, and functional units and enables a much more strict process isolation in the sense that sensitive information never leaks outside the zone. Since the units of the protected zone and its organization are designed with cryptographic constraints, it also provides superior time performance in comparison with equivalent architectures. The design of the protected zone is highly modular, and complies with the design principles of RISC processors, therefore, it can be incorporated into any RISC processor. We also demonstrate that well-know cryptographic algorithms, RSA, ECC, AES can be implemented on embedded processor equipped with the protected zone with superior time performance, efficiency, and high-level security. A similar approach is used in [21] only for AES implementation; however the proposed techniquein [21] cannot easily be extended to more complicated public key algorithms. We provide a complete, generic approach that is readily applicable to any cryptographic algorithm and does not necessitate Assembly language implementation which is essential in [21].

II. Principles and Requirements of Secure and Isolated Execution

Software implementations of cryptographic algorithms are vulnerable to various forms of attacks that can be grouped into two main classes: side-channel [14], [15] and fault-injection attacks [4]. Different countermeasures from circuit- [20] through architectural- [17] to algorithmic-levels [11] have been proposed. It has, however, been well-understood

that ultimate protection against all kinds of attacks seems to be impossible and correct combination of the effective counter-measures need to be deployed for reasonably secure implementations of cryptographic algorithms. In this paper, we deal with architecture-level attacks and countermeasures.

Many cryptographic algorithms utilize lookup tables for fast execution, which makes them vulnerable to cache-based side-channel attacks [7]. Another side-channel attacks that utilizes processor micro-architecture is named as branch-prediction attacks [1]. The main reason that these attacks are effective is the fact that majority of general-purpose processors (including many embedded processors) support multi-tasking and resource sharing as in the cases of cache memories, branch prediction and target buffers. The processes running simultaneously cannot directly access each other's data since the operating system enforces process isolation. However, processes inadvertently (and inevitably to a certain degree) leave data in shared resources (cache memories and branch buffers). Another process cannot directly use or learn the residue data; however, it can make inferences through carefully timed accesses to these shared resources. The residue data in shared resources do not have to be secret or confidential per se; but their presence may say something about the secret that is used to access to them. Naturally, during the execution of cryptographic algorithms, secret keys are used to access lookup tables (hence cache attacks) and to make decisions in the program execution flow (hence branch prediction attacks).

Worse yet, the bugs and flaws in operating system render OS-implemented process isolation ineffective against sophisticated attacks that allow ill-intentioned programs to gain access to secret information through the violation of process isolation. This situation calls for a much stronger, and inevitably hardware-based, mechanism for process isolation. Supporting this claim, major processor manufacturers such as Intel, AMD, and ARM, introduced extensions to their processor cores to fortify the process isolation [6], [8], [2]. The basic principle is making certain parts of memory, of cache, and of TLB used by a process strictly inaccessible by other processes. However, the isolation is still virtual rather than physical since the data from different processes still occupy the shared resources. For instance, the confidential data such as secret keys and temporary variables will still be present in physical memory in certain points of execution. Recently demonstrated cold-boot attacks [10] efficiently recover secret keys used in cryptographic operation.

Therefore, to provide even stronger type of process isolation where the cryptographic algorithms execute free from vulnerabilities against the attacks mentioned, processor architecture needs to provide support for keeping all the confidential information in physically protected zones. The confidential information not only include secret keys but all the intermediate values obtained during the cryptographic computation. For example, an AES block in an intermediate round is also confidential since its compromise may reveal important information on the secret key. Similarly, an elliptic curve point obtained during elliptic curve scalar multiplication needs to be protected, since it is possibly a smaller multiple of the base point which gives away certain bits of the secret integer (possibly private key). Therefore, there is a need for a protected zone where we can keep the confidential information before, during, and after the cryptographic computation. The protected zone includes functional units, a small, protected local memory, a register file that we can use during operations, and some special registers to keep some intermediate variables. In what follows, we explain the components of the protected zone.

- **Functional units** execute the instructions needed in cryptographic computations, which basically implements simple arithmetic/logic operations. Some operations are needed for secure execution of cryptographic algorithms to prevent branch prediction attacks as well as to avoid confidential variables appearing in architectural registers of the processor.
- **Local memory** is used to implement a scratch pad for temporary variables and lookup tables as well as to keep secret keys. The local memory can be implemented on-chip as well as outside of the chip; but the important feature is that it is physically protected and not a part of the memory hierarchy to avoid it from being backed up on higher levels of the hierarchy. The local memory can be thought of a device that responds to access requests to a certain range of memory address, which is especially easy to implement in processors using memory-mapped I/O technique to access peripherals. Its implementation is much easier than a cache memory since placement scheme is straightforward. The cache memory usage is always problematic in cryptographic algorithms and not necessarily as beneficial as a possible local memory so far as the speed is concerned. Some commercially available processors such as graphic processors and Cell architecture [5] also feature local memories. It is important to note that SPE cores in Cell architecture use on-chip local memories in isolation.
- **Registers** are organized as a register file from which the functional units can operate on. The confidential values (secret keys and sensitive temporary values) are kept and operated while they are in these registers. Important feature of these registers is that they are not spilled onto the main memory but to the local memory.
- **Special registers** are used to keep some temporary values during the long-latency cryptographic computations such as multi precision modular multiplication and block cipher round operations.

In the next section, we propose an architecture to realize such a protected execution zone within a general-purpose

processor.

III. General Architecture

The architecture in Figure 1 is proposed to fulfill the requirements of secure and isolated execution of cryptographic algorithms stated in Section II. The base architecture is basically a 32-bit embedded processor core based on Xtensa LX2 architecture [19] that provides basic integer functionality. The architecture is both reconfigurable and extensible. Basic pipeline structure, a register file of 32 32-bit registers and a simple ALU are default resources in what is referred as base architecture whose components are shown in dark in Figure 1. The resources shown in the lightest represent configurable parts, which simply means that developer/designer can choose to add/remove/configure units already available in the Xtensa LX2 architecture. For instance, a 16- or 32-bit multiplier (MUL16/32 in Figure 1), multiply-and-accumulate unit (MAC in Figure 1) can be added to the base architecture. The cache memory size and configuration can also be determined by the designer/developer.

The architecture is extensible in the sense that the designer can add units of her/his own design such as multi-cycle execution units, register files, special registers for multi-cycle instructions, even make the basic RISC pipeline into a multi-issue VLIW processor. This is the feature that we use to realize our protected zone to execute cryptographic operations as shown in Figure 1 (enclosed within the dashed area).

Figure 2 shows the details of the protected zone where we can perform cryptographic operations safely. The organization of the zone is very similar to an ordinary RISC processor core with the exception of 128-bit data path and block cipher unit. Extension register file consists of eight 128-bit registers, which we refer as *cryptographic registers* henceforth, and are used to hold operands during the computation. The execution units, namely integer unit (IU), shifter, and multiplier, are responsible for executing arithmetic/logic operations common in cryptographic computations in an efficient and secure manner. While 128-bit shift and arithmetic/logic instructions are single-cycle, the 128×128 multiplication is a multi-cycle instruction.

The block cipher unit (BCU) is novel in this design and incorporates various operations common in many block cipher algorithms. In the beginning of each round of the block cipher algorithm, the block is in one (or more depending on the block length) of the cryptographic registers. Once the round starts, the block is first transferred into special registers in the BCU. One of the important operations performed in the BCU is *secure* table lookup operation that are employed in many block cipher implementations to accelerate s-box computation. The lookup table is formed inside the local memory in order to avoid cache-based side-channel attacks. RISC-based processors use architectural base registers to compute the exact address of the location of the desired data, which may be directly related to the secret. However, the architectural registers are not safe places to keep confidential information since they are backed up on the main memory; a process that may leak secret information. Therefore, they must be used carefully. A straightforward approach is to reset the architectural register used to keep confidential data after they are no longer needed and before they are spilled to the main memory, which is easy to do in Assembly programming. However, this is not an easy task in higher level language implementations since it is up to the compiler to decide which registers are used in address calculation, which is hard to predict beforehand. We use basically two techniques to reset architectural register's secret content, using high level language constructs that allow inline assembly instructions and defining local variables on specified registers. This way, it is easy to keep track of registers that are used to handle sensitive information to reset them afterward.

Note that certain operations either take multi-cycle or multi-instruction to complete, therefore certain temporary values are kept in special-purpose registers. This resembles multiplication operation that puts the high and low parts of the result in two special-purpose registers, namely `HI` and `LO`, respectively. In order to further operate on the result of a multiplication, instructions such as `mfhi` and `mflo` are used to move the results to the base registers. We adopt the same approach; however, it is required that these special-purpose registers be not saved in the main memory before process switch operation that is supervised by the operating system. Thus, operating system support is necessary in secure and isolated execution to time the context switching in order not to lose data.

IV. The New Instruction Set Architecture

Basic integer arithmetic and logic operations are implemented in protected zone to provide a wide range of cryptographic algorithms with a secure and efficient execution environment. Some of these operations are implemented as simple single-cycle instructions such as integer addition and various shift operations while more sophisticated operations such as wide multiplication are executed in the pipeline with multi-cycle latency. All instructions comply with RISC conventions such as maximum three operands per instruction, simple addressing modes stipulating register-to-register arithmetic, etc. These conventions help keep the data path as simple and regular as possible. For instance, the latency for instruction fetch can be minimized for short and regularly formed instructions.

The implementation of instructions for efficient integer and logic operations have been explained earlier in our earlier work [13]. Therefore, we focus only on explaining new instructions that allow secure execution of cryptographic operations; but only a couple of instructions, which we think are the most representative of the adopted methodology, are explained for space considerations. Firstly, we mention three

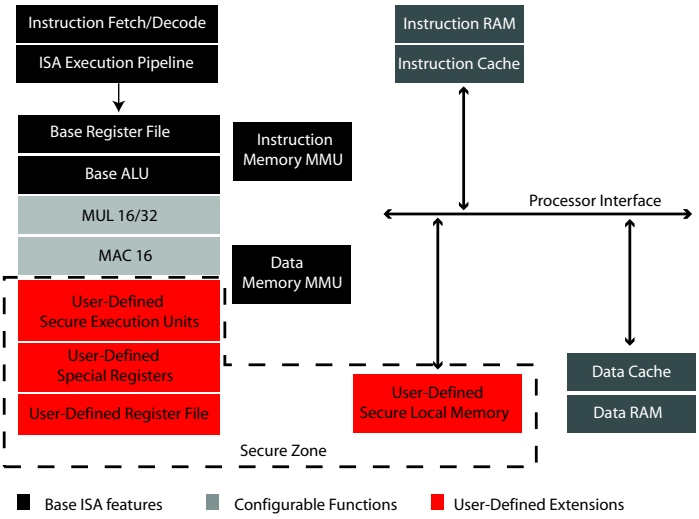

Figure 1. General Architecture

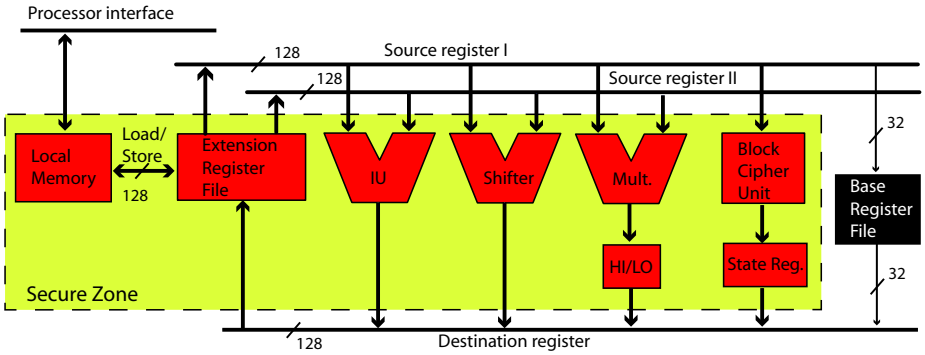

Figure 2. Organization of Protected Zone

special registers that play key roles in secure computations. We use two predicate registers, namely `p0` and `p1` to allow predicated (or conditional) execution of certain instructions. Although the predicated instructions are known, as a novelty we allow arithmetic on predicate registers so that more sophisticated conditions can be evaluated before the completion of an instruction. The other special register, `sbox_in`, is useful in table lookup operations used in implementing s-box computations in block cipher algorithms. Five of the new instructions are given in Table I.

The first instruction `mf_cr2p1` move the most significant bit of cryptographic register `cr` to the predicate register `p1` while the other predicate register `p0` is updated by the old content of `p1` and `cr` is shifted to the left by one bit. The instruction is useful in modular exponentiation and elliptic curve scalar multiplication operations where the secret exponent (or integer) is kept in the cryptographic register and moved to the predicate registers when needed. The next two instructions, `cond_mv` and `cond_mv_c` conditionally

Table I
SPECIAL INSTRUCTIONS

Instruction name	Arguments	Definition
`mf_cr2p1`	`p0, p1, cr`	`p0:=p1;` `p1:=cr[127];`
`cond_mv`	`p1, cr_d, cr_s`	`if p1=1 cr_d:=cr_s;`
`cond_mv_c`	`p1, cr_d, cr_s`	`if p1=0 cr_d:=cr_s;`
`shlcr_ 2sbox_in`	`cr, sbox_in`	`sbox_in:= r[127:96];` shift left `cr` by 32 bits;
`lookup_ table_op`	`addr, base_addr, sbox_in`	`addr:=base_addr + sbox_in[31:24];` rotate left `sbox_in` by 8 bits;

Table II
CLOCK COUNT FOR RSA AND ECC

Algorithm	**Base Architecture**	**Fast on protected**	**Secure protected**
RSA-1024	132,334,584	9,215,168	14,831,132
RSA-2048	NA	66,728,848	107,173,686
ECC-160	5,684,844	2,524,498	4,683,325
ECC-256	21,509,576	3,649,338	7,213,678
ECC-512	160,109,439	16,979,307	33,893,033

move cryptographic register contents from one register to another depending on the value of the predicate register. These instructions are useful again in the exponentiation and elliptic curve scalar multiplication operations where certain operations are performed (e.g. modular multiplication) depending on the current value of the exponent bit which is currently in the predicate register. For instance, in the Montgomery ladder algorithm [11] for exponentiation, the result of the modular multiplication $R0 \times R1$ is assigned either to $R0$ or $R1$ depending on the value of the current exponent bit. The conditional move instructions are useful in performing this assignment without using branch prediction circuit that leaks information about the secret key [1]. By performing secret key-dependent move instruction, the branch prediction attacks are thwarted.

The next two instructions in Table I are used in secure table lookup operations in block cipher algorithms. The instruction `shlcr_2sbox_in` moves the highest 32-bit of `cr` to the special register `sbox_in` while the instruction `lookup_table_op` computes the address of the s-box output value precisely, which allows fetch the desired item from the lookup table securely. The calculated address and s-box output are placed in architectural base registers which need to be properly handled and erased afterward.

V. IMPLEMENTATION RESULTS

We synthesized the protected zone into a simple Xtensa LX2 core and implemented various cryptographic algorithms such as RSA, elliptic curve cryptography (ECC), and AES on the resulting processor. The timing results of an RSA exponentiation and a ECC scalar point multiplication are given in Table II in terms of number of clock cycles. Note that all implementations are done in C language and implementations in Assembly are expected to yield better performance. Note also that both fast and secure versions execute completely in isolated manner while the secure versions are additionally hardened against side-channel (e.g. the Montgomery Ladder is used), cache and branch prediction (local memory is used and key dependent branches are eliminated) attacks. For comparison, our implementations, both fast and secure, greatly outperform another 1024-bit RSA implementation on the same Xtensa processor in [18] that takes 24.32 million clock cycles.

Similarly, we implemented the AES algorithm and the results are given in Table III along with those of other implementations on similar embedded platforms. As seen in the table, our implementation outperforms all the other implementation except for the one in [9], which is a bit-sliced implementation. Our implementation does not use bit-slicing technique, therefore, can work in any mode of operation. A bit-sliced implementation in our architecture would possibly yield a better performance, which we leave as a future work.

Table III
COMPARISON OF AES IMPLEMENTATIONS

Implementation	**Hardware support**	Performance (cycles)
[3] on ARM7TDMI	-	1675
[16] on AMD Opteron	-	2699
[9] on CRISP	Bit-sliced + lookup tables	2203
[9] on CRISP	Bit-sliced + lookup table + bit-level permutation	1222
[18] on Xtensa	-	1400
this work	**on protected zone**	**1334**

We implemented our protected zone and incorporated it into a basic embedded processor core, namely Xtensa by Tensilica. The Tensilica tool chain estimates that the protected zone takes a chip area of $92,924$ equivalent gate count in ASIC realization. The tool chain did not report any penalty in maximum applicable clock frequency. Another important figure that we need to consider is the size of the protected local memory required for software implementations of cryptographic algorithms. It turns out that the required memory sizes are surprisingly low. RSA, ECC, and AES algorithms need 5700 B, 1936 B, and 464 B of scratch pad memory, respectively. RSA memory requirement in fast implementations can be as low as 1860 B at the expense of $17 - 18\%$ deterioration in speed. The secure RSA implementation requires only 2112 B memory space. Considering it is possible to realize as large as 256 KB local memory per each processor in Cell architecture [5], the memory requirements of our implementations are very low.

We also synthesized the design into an FPGA target device, and generated configuration files to program Avnet LX200 board that features Xilinx Virtex-4 type FPGA. This basically means that the reported implementation figures are obtained after placement-and-routing. The number of slices used in the design is reported to be 29707; and 15151 slices of the total number are used to implement the protected zone. No degradation in maximum clock frequency is reported by the synthesis tools; 50 MHz maximum clock frequency is achieved for the design.

VI. Conclusion and Future Work

We designed, implemented and realized a protected zone in an embedded processor that allows efficient, secure, and isolated execution of cryptographic algorithms. We estimated the area overhead of the protected zone for the ASIC implementation. We also provided area usage on an FPGA device after placement-and-routing for the full design including an embedded base processor and protected zone. Since the number and organization of the subsystems in the protected zone are carefully designed, we observed only a moderate area overhead while no deterioration in maximum applicable clock frequency is reported. We outlined the principles of the software implementation of cryptographic algorithms so that resulting executable runs in a secure and isolated manner. Since the protected zone is specifically tailored for the cryptographic application domain, we achieved superior time performance of major cryptographic algorithms compared to both those reported in literature and those in the base processor. The protected zone can be instrumental in implementing a TPM within the microprocessor. For this, we need to investigate secure implementation of cryptographic hash functions whose security constraints are not about the confidentiality of temporary values but about maintaining their authenticity; a task that can easily be performed in our architecture.

Acknowledgment

This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project number 105E089 (TUBITAK Career Award).

References

[1] O. Aciiçmez, Ç. K. Koç, and J.-P. Seifert. Predicting secret keys via branch prediction. In M. Abe, editor, *CT-RSA*, volume 4377 of *LNCS*, pages 225–242. Springer, 2007.

[2] ARM. *TrustZone Technology Overview*. http://www.arm.com/products/security/trustzone/.

[3] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, and S. Marchesin. Efficient software implementation of aes on 32-bit platforms. In Jr. et al. [12], pages 159–171.

[4] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT*, pages 37–51, 1997.

[5] T. Chen, R. Raghavan, J. Dale, and E. Iwata. Cell broadband engine architecture and its first implementation. Website, 2005. http://www.ibm.com/developerworks/power/library/pa-cellperf/.

[6] Intel Corporation. *LeGrande technology preliminary architecture specification*. Intel Publication no. D52212, May 2006.

[7] D. Bernstein. Cache-Timing Attacks on AES. Website, 2005. http://cr.yp.to/papers.html#cachetiming.

[8] Advanced Micro Devices. *AMD64 virtualization: Secure virtual machine architecture manual*. AMD Publication no. 33047 rev. 3.01, May 2005.

[9] P. Grabher, J. Großschädl, and D. Page. Light-weight instruction set extensions for bit-sliced cryptography. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *LNCS*, pages 331–345. Springer, 2008.

[10] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proc. 17th USENIX Security Symposium (Sec '08), San Jose, CA*, July 2008.

[11] M. Joye and S.-M. Yen. The montgomery powering ladder. In Jr. et al. [12], pages 291–302.

[12] B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors. *CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *LNCS*. Springer, 2003.

[13] Ö. Kocabaş, E. Savaş, and J. Groβschädl. Enhancing an embedded processor core with a cryptographic unit for speed and security. In *RECONFIG '08*, pages 409–414, Washington, DC, USA, 2008. IEEE Computer Society.

[14] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[15] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[16] R. Könighofer. A fast and cache-timing resistant implementation of the AES. In T. Malkin, editor, *CT-RSA*, volume 4964 of *LNCS*, pages 187–202. Springer, 2008.

[17] D. Page. Partitioned cache architecture as a side channel defence mechanism. Cryptography ePrint Archive, Report 2005/280, August, 2005. citeseer.ist.psu.edu/page05partitioned.html.

[18] S. Ravi, A. Raghunathan, N. R. Potlapally, and M. Sankaradass. System design methodologies for a wireless security processing platform. In *DAC*, pages 777–782. ACM, 2002.

[19] Tensilica. Xtensa LX2 Embedded Processor Core. Website. http://www.tensilica.com/products/xtensa_LX2.htm.

[20] K. Tiri and I. Verbauwhede. Securing encryption algorithms against DPA at the logic level: Next generation smart card technology. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *CHES*, volume 2779 of *LNCS*, pages 125–136. Springer, 2003.

[21] K. Yumbul and E. Savaş. Efficient, secure, and isolated execution of cryptographic algorithms on a cryptographic unit. In A. Elçi and et al., editors, *SIN*, pages 143–151. ACM, 2009.