Instruction Set Extensions for Secure Applications

Francesco Regazzoni* and Paolo Ienne†

* ALaRI - University of Lugano, CH-6900 Lugano, Switzerland, regazzoni@alari.ch
† School of Computer and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne (EPFL),
CH-1015 Lausanne, Switzerland, {paolo.ienne}@epfl.ch

***Abstract*—The main goal of this paper is to expose the community to past achievements and future possible uses of *Instruction Set Extension* (ISE) in security applications. Processor customization has proven to be an effective way for achieving high performance with limited area and energy overhead for several applications, ranging from signal processing to graphical computation. Concerning cryptographic algorithms, a large body of work exists on speeding up block ciphers and asymmetric cryptography with specific ISEs. These algorithms often mix non-standard operations with regular ones, thus representing an ideal target for being accelerated with dedicated instructions. Tools supporting automatic generations of ISEs demonstrated to be useful for algorithm exploration, while secure instructions can increase the robustness against side channels attacks of software routines. In this paper, we discuss how processor customization and the relative tool chains can be used by designers to address security problems and we highlight possible research directions.**

I. Introduction

Smart devices are already pervading several aspect of our lives, and they will certainly do even more in the near future. Wearable and implantable systems will be used for monitoring our health parameters. Smart mobile devices will control our houses, enabling a more efficient use of energy resources. Intelligent cyber-physical systems will handle all the operations of the smart grid and they will guarantee the safety of our transport systems. The “heart” of our entertainment will be our smart phone, which will be able to access our contents saved on the cloud from wherever we are. Our smart phone will also be used for accessing our bank account and handling all the payments. All of this can be achieved only with extremely efficient computational elements, characterized by a minimal energy consumption, since most of the devices will be battery operated, and achieving a level of performance compatible with the requirements of the target applications.

However, these applications will handle extremely sensitive information (such as the ones related with our health conditions or our bank account) or will help to manage critical infrastructure (such as the smart grid or the transport systems). For this reason, future smart devices must include, at least, the security features needed to guarantee the safeness and the reliability of their operations. Required security functions might include private key encryption, maybe carried out using a lightweight cipher, public key encryption, and authentication of transmitted data and eventually of the transmitting device. Furthermore, considering the possibly hostile environment where systems can be deployed, devices should provide resistance against physical attacks and tampering.

Security can be added to embedded devices by means of specific software libraries or by means of dedicated hardware accelerators. The first approach is very flexible, but for extremely constrained devices can be too costly in terms of memory occupation or energy consumption and the performance can may be sufficient to meet the application requirements. Furthermore, certain security threats may not be defeated using software solutions only. The second approach allows to achieve better performance and often better security, but is not flexible and often requires an excessively large area occupation. Additionally, depending on the frequency and the amount of data which needs to be moved to the accelerator, the delay of the transfer could overshadow the speedup of the dedicated hardware.

An appealing approach to guarantee high performance to cryptographic routines while limiting the cost and area overhead consists in extending the base instruction set of embedded processors with a number of instructions dedicated to security. Processors customization is an active area or research which aims at reducing the gap between the application requirements and the features offered by standard processor. It consists in specializing processors for specific functions, where the approach one-fits-all, usually followed while designing general purpose processors, is not the best option. Processor customization is extremely suitable for designing embedded devices or cyber-physical systems, as they are often very specialized.

Processor customization for cryptography was firstly introduced by Nahum et al. [17]. The authors, instead of implementing whole algorithms using dedicated hardware, propose to improve the performance of cryptographic software by adding to a standard RISC instruction set few instructions to deal with operations on sub-wordsize units, operations on super-wordsize units, and operations on groups other than that on integers. Following this seminal work, researchers and industry proposed and implemented a number of possible extensions aiming at accelerating block ciphers and asymmetric algorithm. Furthermore, over the years, processor customization and the related supporting tool chains were used also to increase the robustness of cryptographic software against power analysis and timing attacks.

This paper reports on innovations that have followed since 1995 in the domain of Instruction Set Extensions for security, with the intention of exposing the community to the possible use of processor customization for addressing current and

978-3-9815370-6-2/DATE16/©2016 EDAA

future security requirements of embedded systems and for addressing the needs of the next generation of cyber-physical systems.

II. Customizable Processors

A customizable processor is essentially a basic processor architecture, the custom processor template, which includes all the needed interfaces to plug a number of additional instructions. These instructions are defined by the designer and are integrated into the base processor architecture in the form of functional units. The final result of the customization is a processor tailored to a specific application. The custom instructions can then be instantiated in software routines exactly as the instructions of the base architecture.

An example of extensible processor, in this case having a two-inputs one-output functional unit (AFU), is depicted in Figure 1. As it can be seen, the custom logic blocks are located in the datapath of the processor and they have direct access to the register file and to the data memory. The amount of I/O of the register file available for the functional unit can be limited. However, this limitation can be overcome by using multicycle reads/writes from/to the register file. With this approach, it is possible to squeeze several operands into the two input-one-output register file [20].

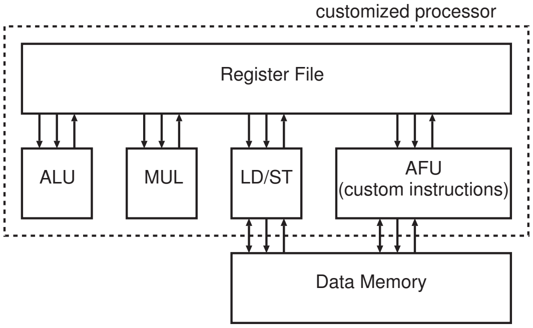

Fig. 1: Extensible processor with a custom two-inputs one-output functional unit.

Customizable processors usually include also a tool chain for the development of user applications and, in some case, for the automatic generation of custom instructions. An example of these tool chains is depicted in Figure 2. The starting points are the HDL code of a customizable processor and the software application on which the processor will be tailored (in this case, a cryptographic routine). The tool for the automatic identification of ISEs (ISE Extractor in the figure) will identify several potential instructions that could maximize or minimize a specific optimization parameter. The designer, considering all the constraints of the target application, will select one or more custom instruction to be added to the processor instruction set. The tool will generate the HDL code of the new instructions as well as the new software application which already instantiate the newly added custom instructions. The whole code of the processor is then synthesized and placed and routed, while the software application is recompiled. Compilers included in the customizable processor tool chains are often already capable of recognizing a certain number of custom instructions.

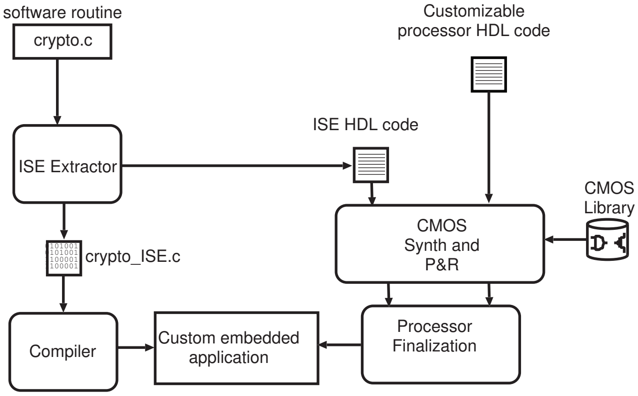

Fig. 2: Automatic Identification of Instruction Set Extensions Design Flow.

A more detailed view of the automatic generation of instruction set extension is visible in Figure 3, which reports the start and the end point of the acceleration of software routine computing the AES algorithm. We assume that the designer decided to extend the processor with an instruction performing both the key addition and the look up table of the non linear transformation in a single step. On the left side of Figure 3, it is possible to see the original code written in C language, where the key addition and the non linear transformation are implemented using only the instructions available in the base architecture. The right side of the figure reports the HDL code of the custom instruction (on the top) and the updated software routine (on the bottom) instantiating the new instruction, called in this case *Instr_1*.

Several customizable processors are available on the market [2], [10], [5], [22], each of them including a dedicated tool chain. The specific way in which the instruction have to be integrated into the base processor and the specific support depend on the vendor. Nios II [2] is a soft core processors included in the tool chains of Altera FPGAs. The processor can be easily customized with dedicated instructions. When custom instructions are added to the Nios II base architecture, each custom operation is assigned a unique selector index. The selector index, which is determined during the instantation of the hardware, allows the software to specify the desired instruction. The processor supports up to 256 custom operations.

CorExtend [10] supports the implementation of a small number of application-specific instructions tightly coupled to the execution unit of MIPS cores, using an interface external to the core itself. These instructions can operate on a general-purpose register, immediate data specified by the custom instruction, or on local data stored within the user defined instruction block. The destination may be a general-purpose register or a local data block included in the dedicated instruction. The provided support allows to implement instructions which can be complied in one or multiple clock cycles.

Original AES routine

```
// Calculate AES (plaintext, key)
int AES(int pt, int key) {
int S[256] = { 0x63, 0x7C, ...} // declare the Sbox
int result = 0; // initialize the result
... // other instructions
 // calculate plaintext ˆkey
asm volatile(”l.xor %[out1],%[in1],%[in2]”:[out1]”=r”(pt):[in1]”r”(pt),[in2]”r”(key));
result = S[pt];  // calculate the non linear transformation
... // other instructions
return result; }; // return the result
```

HDL code of the new instruction

```
... // other instructions
internal_byte <= v_in_op1 xor key_byte;  // compute the key additon
// instantiate the sbox component and compute the non linear transformation
inst_SBox : sbox
port map (InpxDI => internal_byte,
    OupxDO => v_out );
... // other instructions
```

Updated software routine

```
// Calculate AES (plaintext, key)
int AES(int pt, int key) {
int result = 0; // initialize the result
... // other instructions
Instr_1(pt, key, result); // calculate pt ˆkey and the non linear transformation
... // other instructions
return result; }; // return the result
```

Fig. 3: Example of ISE generation. On the left, the original code which allocated the memory for the Sbox and compute the key addition and the non linear transformation with the instructions of the base architecture. On the right, the custom instructions. On the top, the HDL code of the instruction, on the bottom, the updtaded software routines which instantiates the newly generated instruction.

Tensilica [5] is a RISC processor family which is the base of the processor generator tool chain of Cadence. The tool chain allows the designer to easily integrate in the base processor or in DSP new instruction set extensions.

Customizable processors were also proposed and widely used in academia. Possible examples are OpenRISC [21], SPARC V8/Leon [25] and 8 bit AVR microcontrollers [26].

III. Instruction Set Extension for Increasing Performance

The most common parameters which are optimized using custom instructions are clock cycle count and memory footprint. The performance increase however should be achieved with a minimal area overhead. In the case of security applications, Instruction Set Extensions were explored to speed up several categories of algorithms, including symmetric ciphers, asymmetric cryptography, and hash functions.

Symmetric ciphers often include unconventional rotations and a non linear transformation, usually realized as a look-up-table. The introduction of few dedicated instructions to support fast substitutions, permutations, rotations, and modular arithmetic can lead to a speed-up larger than 50% [4] (which can be even larger if the base architecture does not include specific instructions for rotations).

In the context of custom instruction, the cryptographic algorithm which was most extensively studied was probably the *Advanced Encryption Standard* (AES) [18]. AES is a block cipher which supports key sizes of 128, 192 and 256 bits, and block size of 128 bits. The round function is composed of four transformations: *ShiftRows* that cyclically shifts left the bytes in the last three rows of the state with different offsets, *SubBytes (or S-box)* the non-linear transformation, *MixColumns* that compute a modular multiplication between the columns of the state and a given polynomial, and finally *AddRoundKey* that adds a round key to the state. Each round key consists of 128 bits produced by a key schedule routine. The round is iterated a specific number of times depending on the key size. Decryption is similar to the encryption process and uses the same basic transformations, but reversed.

Most of the work devoted to the acceleration of the AES algorithm were targeting 8-bit [26] or 32-bit [3], [24] architectures. As in the general case of symmetric ciphers, also for AES the most relevant operations to be accelerated are substitutions and modular multiplications (rotations of the AES algorithm can often be efficiently implemented with instructions of the base instruction set). Custom instructions for computing the S-box can be easily implemented by extending the processor with a small look-up table storing the results of the computation. The S-box of the AES algorithm is computed on 8 bits at a time, which is often also the size of the data path of small microcontrollers. For these processors, the *sbox* instruction would receive one operand of 8 bits as input, and return an 8-bit output as results of the nonlinear transformation. For processors with larger data path, for instance 32-bit, the same custom instruction would be implemented instantiating four look-up-tables. In this case, the *sbox* instruction would receive one operand of 32 bits, the whole line of the state matrix, and perform the nonlinear transformation on all the 4 bytes of the word in parallel.

In reality, however, the tendency is to try pack as many operations as possible into a single instruction. As a result, instructions accelerating the nonlinear transformation are often combined with the computation of the MixColumns (or one portion of it) and sometimes also with the key addition. Nevertheless, simpler custom instructions are also proposed, mainly for the computation of the last round, which does not include the MixColumn step.

An example of AES instruction set extension for an 8-bit microcontrollers is reported by Tillich and Herbst [26]. The authors propose two instruction variants for AES encryption, called *AESENC*, having the same format of the integer multiplication instruction *MUL* of the basic AVR architecture. The instruction computes the S-box and a multiplication with a specific constant for performing the MixColumn. The constant depend on the instruction variant: thanks to the symmetry of the MixColumn, two variants of this instruction are sufficient to transform the whole AES state. These instructions make it possible to perform all the transformations of a single AES round on two bytes of the state with two invocations. An additional *AESSBOX* instruction is provided to perform the nonlinear transformation in the last round. The key addition and the ShiftRows are implemented using instructions from the base AVR instruction set. Dedicated instructions for decryption are implemented following the same principle used for designing the ones for the encryption. The amount of clock cycles required to perform an encryption are slightly less then half compared with the software implementation realized using the base instruction set. Similarly, the memory footprint is also halved. The area overhead is approximately one third of the one needed to implement a compact coprocessor.

Bertoni et al. [3] and Tillich et al. [24] presented examples of custom instructions for supporting AES in 32-bit processors. Both articles propose byte-oriented (operating on a single byte) and word-oriented (operating on the whole processor word, namely 4 bytes in parallel) custom instructions. Bertoni et al. proposed one instruction to perform both the S-box and the MixColumn in the same instruction. The byte oriented version firstly selects the correct byte from the processor word, then applies the S-box transformation to it, and finally computes the four different contributions of that byte to the MixColumn. The correctness of the final result is ensured by an appropriate reordering of the output bytes, which is done automatically and accordingly to the definition of the MixColumn. A byte-oriented instruction performing only the S-box of a single byte is provided for the last round of the AES algorithm. In the word-oriented approach, the MixColumns is computed for the whole column in a single clock cycle. Also, in this case, a dedicated instruction to perform four S-boxes in parallel is proposed. Clock cycles were estimated to be approximately one third of those of a software implementation using the base instruction set. Tillich et al. [24] propose to use two different instructions, one to perform the S-box and one to perform the MixColumn. These two steps were kept separated also in the word-oriented implementation of the custom instructions. Here too, the amount of clock cycles needed to compute the encryption is significantly reduced compared to a software implementation using only the base instruction set of the target processor.

Custom instructions were designed also to accelerated the arithmetic operations in finite fields. Großschädl and Savas [8], for instance, proposed five custom instructions to accelerate arithmetic in $GF(p)$ and $GF(2^m)$. The instructions are designed to compute the following operations, respectively: multiply two 32-bit integers, double the product, and accumulate it into a target register; compute the sum of two unsigned integers and accumulate it into a target register; shift the values in a target register to the right; perform a multiplication over $GF(2)$ and store the result into a target register; and finally multiply two binary polynomials and add the results to a target register. Reported results measured on an extended MIPS core showed that an elliptic curve scalar multiplication can be performed approximately six times faster than a software implementation on the standard version of the same processor.

Custom instruction designed for one specific algorithm can be useful also for increasing the performance of another one. An example of this is acceleration of the AES algorithm using three of the five instructions previously described and designed to improve field arithmetic [23]. Although not specifically designed for that purpose, the use of these instructions allows performance gains of up to 25%. Interestingly, this speedup is obtained at no additional hardware cost on processors which already feature these instructions.

Custom instructions were also designed to accelerate hash functions. Recent work [7], [6] investigated for instance the benefits of custom instructions for improving the performance of the five SHA-3 final round candidates [19]. The authors demonstrated that hash function would benefit significantly from the presence of finite state machines for quick generation of the addresses needed to perform the permutations, from the integration of look-up-tables into the data path, and from the extension of computational units to support rotations and matrix multiplications. Experimental results carried out using a 16-bit microcontroller architecture demonstrated that using 10% of additional core area, it is possible to achieve an increase of the speed ranging from 21% to 7x, depending on the algorithm, while memory requirements reduced by approximately of 40% on average.

Finally, custom instruction dedicated to cryptography were included also into general purpose processors. The most relevant example are probably the AES-NI [11] instructions, included in the Intel processors since the Westmere architecture [12]. The AES-NI instruction set is composed of six instructions that to compute several portion of the AES algorithm. More in details, two instructions are accelerating the encryption (*AESENC* which performs ShiftRows, SubBytes, MixColumns and AddRoundKeym and *AESENCLAST* which is used in the last round of the algorithm thus skipping the MixColumn), two instructions accelerated the decryption (*AESDEC* and *AESDECLAST*), and two instructions are designed for key generation (*AESKEYGENASSIST* and *AES-IMC*). Performance gain ranges between 2 to 10 times over pure software solutions.

IV. Instruction Set Extension for Side Channel Resistance

Instruction set extension can be also designed to achieve goals different than increasing performance or minimizing memory footprint. An appealing application for custom instruction is to use them to harden software routines against physical attacks. Physical attacks are a particular class of attacks in which the adversary attempts to retrieve the secret information stored in or processed by a device by exploiting the weaknesses the implementation rather than attacking the mathematical structure of the cryptgraphic algorithms themselves.

Several physical attacks proposed so far exploit the information leaked during the computation through a physical observable quantity, such as the time needed for computation [14] or the power consumption [13]. Due to the pervasive diffusion and the potentially hostile environment where they are deployed, embedded and cyber-physical systems represent an appealing target for physical attacks.

Power analysis attacks is probably the most successful physical attack presented so far. It infers the secret key of a cryptographic algorithm by exploiting the difference between the power consumed by a device during the computation. To date, a perfect countermeasure against these type of attacks does not exist. However, designers can significantly complicate the task of the adversary. Software countermeasures proposed so far consists in hiding the exploitable information in the power traces, by for instance adding random delays or instructions [15], or in computing the encryption on randomized data [16] (this approach is called masking). However, software countermeasures cause significant performance degradation. Furthermore, achieving good protection relying only on software is an extremely difficult task [25].

Custom instructions, combined with power analysis resistant logic styles [27], [21] could help achieving this goal [21], as depicted in Figure 4. Tools originally designed to extract ISEs for performance can be easily adapted to a different objective, the minimization of information leakage. The selection of the particular instructions is then carried out with the ultimate goal of minimizing the possibility of success for a power analysis attacker. Once selected, the custom instructions are synthesized using logic gates designed to be resistant against power analysis and placed & routed in a careful way to avoid unbalanced wires. The base processor is synthesized using the usual CMOS design flow. Processors and secure ISEs are finally integrated and the updated software routine, already instantiating the secure instructions, can be compiled with the usual compiler.

Custom instructions are useful also to increase the resistance of software countermeasures, combining them with hardware ones. For instance, masking can be implemented in a more robust and efficient way when dedicated instructions for mask generation and handling are added to the base processor and implemented in protected logic style [25].

Timing attacks, which exploit the timing difference during computation, can also be minimized using dedicated instructions. To defeat these attacks, a designer should ensure that the computation of exploitable portion of a cryptographic algorithms are carried out in constant time. Custom instructions for achieving this purpose were proposed in the past. The most relevant example of them are the already mentioned AES-NI [11]. These instruction allows to compute encryption and decryption without the need for software look-up-tables, therefore lowering the risk of timing attacks.

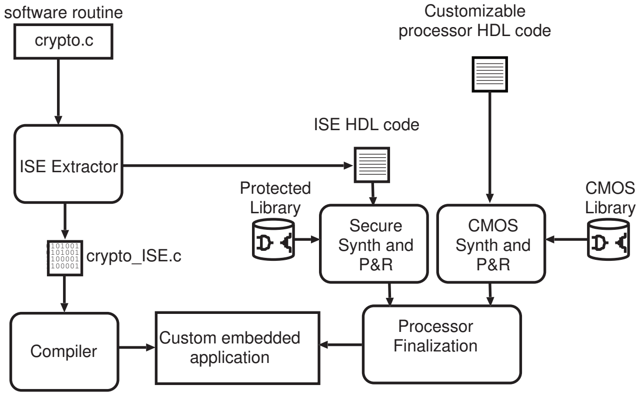

Fig. 4: Automatic Identification of Instruction Set Extensions for DPA resistance. The tools and the process of selection of instructions is similar to the one carried out when designing ISEs for improving performance, however, the objective function in this case is the minimization of leakage of information through power. Custom instructions in this case have to be synthesized and placed & routed using a protected logic style and not with the regular CMOS design flow.

V. Lesson Learned and Future Challenges

Since the first proposal of instruction set extension for secure applications, processor customization has been an active topic of research for the cryptographic community. Several processors, also general purpose ones, include today instructions specifically designed for computing cryptographic primitives. These instructions are often also suitable for increasing the resistance against physical attacks. Design tools originally realized for efficiently selecting ISE have been successfully used also for improving design variables other that performance, such as resistance against power analysis attacks.

However, processor customization and the related tool chains can further help the security community in several aspects. ISEs accelerating lightweight cryptographic algorithms are useful for providing security to processors used for wearable and medical applications at an extremely low cost. Embedded processor can be augmented with robust security primitives implemented in hardware. Several processors, for instance, already include instructions for the generation of random numbers, but other functionality are hard to implement in pure software can be integrated as well. Yet, ISEs seems the best approach for adding security to applications where the resources are very limited and the used algorithms are known well in advance (as is the case of cyber-physical systems). In this context, automatic ISE would enable an aggressive joint optimization of security and non-security functionalities. Processor customization would bring also further advantages such has low cost robustness against physical attacks,

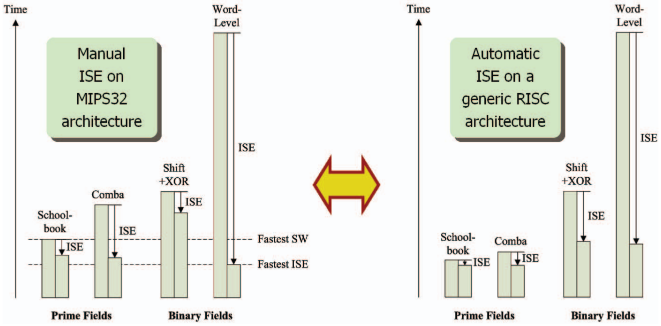

Fig. 5: Automatic ISE as exploration tool: compared to the manual situation, the automatic exploration reverted the relative performance of different algorithms.

An additional application for automatic selection of ISE is algorithm exploration. The availability of ISE can have a dramatic impact on the performance of different algorithmic choices implementing identical or equivalent functionality. System designers need fast feedbacks on the possibility of improving a software routine with dedicated instructions. Automatic ISE, even without predicting speed-ups as precisely as detailed simulation can nor trying to achieve the ulitmate ISE implementation, is able to show the trends that the designer should follow. This approach was successfully used to demonstrate that, in elliptic curve cryptography, ISE can reverse the relative performance of different algorithms implementing the same operation [9], as depicted in Figure 5. A similar approach would be useful also during the selection process of an algorithm, such as the current Caesar context [1]. For cryptographers, tools to automatically discover ISEs may be an invaluable window on the computer engineering potentials of different solutions, promising a future of new standards which will be best compromises between cryptographic robustness and efficient implementability in hardware and software.

References

[1] "Caesar: Competition for authenticated encryption: Security, applicability, and robustness," 2013, http://competitions.cr.yp.to/caesar.html.

[2] Altera, *Nios II Custom Instruction User Guide*, Nov. 2015. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_nios2_custom_instruction.pdf

[3] G. M. Bertoni, L. Breveglieri, F. Roberto, and F. Regazzoni, "Speeding up AES by extending a 32 bit processor instruction set," in *Proceedings of the IEEE 17th International Conference on Application-specific Systems, Architectures and Processors*, ser. ASAP '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 275–282.

[4] J. Burke, J. McDonald, and T. Austin, "Architectural support for fast symmetric-key cryptography," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS IX. New York, NY, USA: ACM, 2000, pp. 178–189.

[5] *Tensilica Customizable Processor IP*, Cadence Inc., 2015. [Online]. Available: http://ip.cadence.com/ipportfolio/tensilica-ip

[6] J. Constantin, A. Burg, and F. K. Gurkaynak, "Instruction set extensions for cryptographic hash functions on a microcontroller architecture," in *Application-Specific Systems, Architectures and Processors (ASAP), 2012 IEEE 23rd International Conference on*, July 2012, pp. 117–124.

[7] ——, "Investigating the potential of custom instruction set extensions for sha-3 candidates on a 16-bit microcontroller architecture," Cryptology ePrint Archive, Report 2012/050, 2012, http://eprint.iacr.org/.

[8] J. Großschädl and E. Savas, "Instruction set extensions for fast arithmetic in finite fields GF(p) and gf(2^m)," in *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, ser. LNCS, M. Joye and J. Quisquater, Eds., vol. 3156. Springer, 2004, pp. 133–147.

[9] J. Großschädl, S. Tillich, P. Ienne, L. Pozzi, and A. K. Verma, "Combining algorithm exploration with instruction set design: A study in elliptic curve cryptography," Munich, Mar. 2006, pp. 218–23.

[10] *MIPS32 M5150 Processor Core Family Software User's Manual*, Imagination Technologies Ltd, Jul. 2014. [Online]. Available: https://imagination-technologies-cloudfront-assets.s3.amazonaws.com/documentation/MD00980-2B-M5150-SUM-01.02.pdf

[11] Intel, *Intel Advanced Encryption Standard (AES) New Instructions Set*, 2012. [Online]. Available: https://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf

[12] ——, *Intel 64 and IA-32 Architectures Software Developer's Manual*, 2015. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developers-manual.pdf

[13] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," M. Wiener, Ed. Berlin: Springer, Aug. 1999, vol. 1666, pp. 398–412.

[14] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," N. I. Koblitz, Ed. Berlin: Springer, Sep. 1996, vol. 1109, pp. 104–13.

[15] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, ser. Advances in Information Security. New York: Springer, 2007.

[16] T. S. Messerges, "Securing the AES finalists against power analysis attacks," in *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, ser. LNCS, B. Schneier, Ed., vol. 1978. Springer, 2000, pp. 150–164.

[17] E. Nahum, S. O'Malley, H. Orman, and R. Schroeppel, "Towards high performance cryptographic software," in *Architecture and Implementation of High Performance Communication Subsystems, 1995. (HPCS '95), 1995 Third IEEE Workshop on the*, Aug 1995, pp. 69–72.

[18] National Institute of Standards and Technology (NIST), "Announcing the Advanced Encryption Standard (AES)," *http://www.nist.gov/*, November 2001.

[19] "SHA-3 competition," National Institute of Standards and Technology (NIST), 2012. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/

[20] L. Pozzi and P. Ienne, "Exploiting pipelining to relax register-file port constraints of instruction-set extensions," in *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, San Francisco, Calif., Sep. 2005, pp. 2–10.

[21] F. Regazzoni, A. Cevrero, F.-X. Standaert, S. Badel, T. Kluter, P. Brisk, Y. Leblebici, and P. Ienne, "A design flow and evaluation framework for DPA-resistant instruction set extensions," C. Clavier and K. Gaj, Eds. Springer, Sep. 2009, vol. 5747, pp. 205–19.

[22] *DesignWare ARC Processor Cores*, Synopsys Inc., 2015. [Online]. Available: http://www.synopsys.com/IP/PROCESSORIP/ARCPROCESSORS/Pages/default.aspx

[23] S. Tillich and J. Großschädl, "Accelerating AES using instruction set extensions for elliptic curve cryptography," in *Computational Science and Its Applications - ICCSA 2005, International Conference, Singapore, May 9-12, 2005, Proceedings, Part II*, ser. LNCS, O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan, Eds., vol. 3481. Springer, 2005, pp. 665–675.

[24] S. Tillich and J. Großschädl, "Instruction set extensions for efficient aes implementation on 32-bit processors," in *Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems*, ser. CHES'06. Springer-Verlag, 2006, pp. 270–284.

[25] ——, "Power analysis resistant AES implementation with instruction set extensions," P. Paillier and I. Verbauwhede, Eds. Berlin: Springer, Sep. 2007, vol. 4727, pp. 303–19.

[26] S. Tillich and C. Herbst, "Boosting AES performance on a tiny processor core," in *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, ser. LNCS, T. Malkin, Ed., vol. 4964. Springer, 2008, pp. 170–186.

[27] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," Paris, Feb. 2004, pp. 246–51.