A Side-channel Attack Resistant Programmable PKC Coprocessor for Embedded Applications

Nele Mentens, Kazuo Sakiyama, Lejla Batina, Bart Preneel and Ingrid Verbauwhede
Katholieke Universiteit Leuven, ESAT, SCD/COSIC
Kasteelpark Arenberg 10
3001 Heverlee
Belgium
Email: {Nele.Mentens,Kazuo.Sakiyama,Lejla.Batina,Bart.Preneel,Ingrid.Verbauwhede}@esat.kuleuven.be

***Abstract*—This paper describes the design of a programmable coprocessor for Public Key Cryptography (PKC) on an FPGA. The implementation provides a very broad range of functions together with countermeasures against Side-Channel Analysis (SCA) attacks. The functions are implemented in a hierarchical manner, where all levels are accessible by the user. This makes the coprocessor very flexible and particularly suitable to be used in embedded environments where the border between hardware and software needs to be decided depending on the application. Especially for RSA, the resulting implementation on an XC3S5000 FPGA, from the low-cost Spartan series of Xilinx, shows comparable performance figures compared to the state-of-the-art in PKC coprocessors.**

I. Introduction

Public Key Cryptography (PKC) is the basis for security in digital information systems. Key establishment and digital signatures are used to ensure confidentiality, authentication and data integrity. These services are indispensable in network applications such as e-mail, e-commerce and e-banking, but PKC also provides embedded security in mobile applications and credit cards.

Depending on the application, a suitable implementation platform needs to be chosen. For some systems a general purpose microprocessor is satisfactory, while others achieve high performance through cryptographic coprocessors in hardware. Examples of high performance applications are ATMs, trusted computing platforms and biometric devices.

When very high performance is required or when a high volume of coprocessors is needed, ASICs are chosen as implementation platforms. In this case, the reconfigurability of FPGAs is only used for prototyping. However, because of the efforts of FPGA manufacturing companies, the performance gap between ASICs and FPGAs becomes small enough to introduce the trend that FPGAs are more and more used as end products. Especially for low-volume designs FPGAs are very interesting target platforms. FPGAs come with different characteristics and prices. Aiming at cheap products, Xilinx introduced its low-cost Spartan series. The cost of a Spartan FPGA ranges from 20 to 300, but its performance is far below the latest Virtex FPGAs of Xilinx [4]. Because our goal was to develop a cheap FPGA implementation, we used a Spartan for our coprocessor. This made the challenge for competitive performance figures bigger, but nevertheless resulted in an implementation that is comparable to the state-of-the-art in PKC coprocessors.

The most widely used standard for PKC is RSA, which was invented by Rivest, Shamir and Aldeman in 1978 [18]. However, because of its much shorter key lengths, Elliptic Curve Cryptography (ECC), outperforms RSA in silicon area and speed [15], [9]. For this reason, many resource restricted applications are evolving towards ECC. From these observations, it becomes clear that there is a need for PKC coprocessors supporting both RSA and ECC. Whereas RSA imposes operations in prime fields, ECC is usually performed using either prime fields or binary extension fields. Our coprocessor provides RSA as well as ECC, where ECC is supported in both fields.

In the mid 1990s, Kocher *et al.* published some Side-Channel Analysis (SCA) attacks [12]. Because FPGA implementations have been repeatedly shown to be side-channel susceptible, countermeasures are indispensable in our PKC coprocessor [17], [22], [24]. For both RSA and ECC, we provide countermeasures against SCA.

To provide maximum flexibility, *e.g.* for programming new algorithms, the user should have access to all operations used by RSA and ECC. This is achieved by implementing the control logic in a hierarchical manner, where all levels in the hierarchy are available for the user. To reduce the complexity of the control logic we use a modular approach: the lowest-level functions in the hierarchy are reused by many higher-level functions. Providing all functions in hardware gives performance and security advantages compared to software approaches like instruction set extensions, where security depends on hard-to-predict branch delays and cache effects.

In summary, this paper introduces an FPGA implementation of a programmable PKC coprocessor. It provides RSA and ECC as well as all underlying field operations and countermeasures against SCA attacks. This results in a very flexible solution providing a broad range of functions. The hierarchical approach keeps complexity under control, which allows the design to be implemented on a low-cost Spartan FPGA.

The paper is organized as follows. Section 2 provides some theoretical background on RSA and ECC. Section 3 gives an overview of the state-of-the-art in PKC coprocessors. In Sect. 4 the implementation of the coprocessor is described, while

Sect. 5 lists the performance and area results and compares our design to the state-of-the-art in PKC coprocessors. Finally, Sect. 6 concludes the paper.

II. Public Key Cryptography

In this section, we elaborate on the theoretical background that is needed to understand the algorithms that are implemented in our coprocessor. Sect. II-A and II-B elaborate on RSA and ECC (in GF(p) and GF(2^n)), respectively. In Sect. II-C some countermeasures for Side-Channel Analysis (SCA) attacks are introduced.

A. RSA

In the RSA standard for PKC [18], the private key of a user consists of two large primes p and q and an exponent d. The public key consists of a pair (N, e), where $N = p \cdot q$ is the modulus (at least 1024 bits) and an exponent e is such that $e = d^{-1} \bmod \lambda(N)$. The corresponding p, q and d are kept secret. To encrypt a message M the user computes $C = M^e \bmod N$ and decryption is described by $M = C^d \bmod N \equiv M^{1+k\varphi(N)} \equiv M \bmod N$. The previous equality follows from Fermat's theorem and the fact that $\lambda = lcm(p-1, q-1)$. The RSA function is the modular exponentiation with the public exponent e and the private exponent d is referred to as the trapdoor to invert the function.

By means of the Chinese Remainder Theorem (CRT), the speed for the RSA decryption scheme can be increased up to 4 times [10]. This possibility is very attractive for practical applications.

B. ECC

In ECC [15], [9], the equivalent operation of the modular exponentiation in RSA is point multiplication, which multiplies a point P on an elliptic curve with a scalar k, resulting in another point on the curve $Q = kP$. The scalar serves as the key, while the coordinates of the points contain the data.

For cryptographic purposes, elliptic curves are defined in finite fields. The most commonly used fields are prime fields (GF(p)) and binary extension fields (GF(2^n)). Point multiplication is performed by repeated point doublings and point additions. These point operations consist of operations in the underlying finite field.

C. Countermeasures against SCA Attacks

The first step in preventing SCA attacks, in particular power analysis attacks, is implementing countermeasures against Simple Power Analysis (SPA). Using an SPA attack an adversary tries to extract secret information by analyzing a single power trace of the implementation while it is executing an operation. The most important action in securing an implementation against SPA attacks, is making the power graphs look indistinguishable, even if different secret information is processed. This includes removing conditional branches.

To prevent Differential Power Analysis (DPA) attacks [11], [12], which use many power consumption traces together with some statistical analysis, the RSA exponent is "blinded". The blinded exponent d' is obtained using $d' = d + \varphi \cdot r$, where r is a random number (of typically 20 bits) and φ is the Euler totient function of the modulus N. More details on exponent blinding are given in [11], [12], where it is proven that $g^d = g^{d'}$. This property makes it possible to choose a different r for each exponentation, changing the exponent without changing the intended result. Because DPA is based on power consumption data for many executions of the algorithm using the same key, exponent blinding succeeds in protecting the cryptosystem against first-order DPA attacks.

For ECC, resistance against DPA attacks can be achieved by key blinding, $k' = k + O \cdot r$, where k' is the blinded key, O is the order of the point and r is a random number (of typically 20 bits). Because kP and $k'P$ always result in the same point on the elliptic curve, this method is effective against first-order DPA attacks when the random number is changed for every execution of the point multiplication. In addition, we implemented another precaution for preventing DPA attacks: Point randomization exploits the fact that the Z-coordinate can be chosen randomly when using projective coordinates [5].

III. Previous Work

Goodman and Chandrakasan proposed a Domain-Specific Reconfigurable Cryptographic Processor (DSRCP) in [6]. The instruction set definition of the DSRCP was dictated by the IEEE 1363 Public Key Cryptography Standard document [8]. The DSRCP performs a variety of algorithms ranging from modular integer arithmetic to elliptic curve arithmetic. The various complex modular arithmetic operations are implemented using microcode, while simple operations are implemented directly in hardware. Multiplication is performed using Montgomery multiplication. For a detailed survey on finite fields multipliers and processors for PKC see [1]. The DSRCP processor is fabricated in a 0.25-μm CMOS technology. The goal of this coprocessor was to achieve an energy-efficient implementation.

Schaumont and Verbauwhede introduced an elliptic curve processor over GF(2^n) in [20], [21]. The architecture has a layered structure with the layers corresponding to the operations described in the security pyramid. The authors propose a language and simulation environment that allows to explore the design of security domain specific processors at a high abstraction level.

Bednara *et al.* gave a comparison of several ways for hardware implementation of elliptic curve cryptosystems in [2]. They especially focused on FPGA based implementations.

There exists also some related work on so-called dual field ECC, which deals with processors that can support both type of fields. Wolkerstorfer [23] proposed a single arithmetic unit that supports addition and multiplication for prime and binary fields. A scalable dual-field ECC processor is described in the work by Satoh and Takano [19].

An example of the utilization of CRT is presented by Grosschädl in [7]. In this paper the multiplier architecture of an RSA chip is presented. The multiplier datapath is reconfigurable to execute either one 1 024 or two 512 bit modular exponentiation in parallel.

The contribution of our paper is that all levels in the hierarchy of instructions are available to the user, which makes the processor completely programmable. Different from the work mentioned above, we implemented many countermeasures against SCA attacks. The FPGA resources were used in an optimal way to provide maximal performance and security on a low-cost FPGA.

IV. Hardware implementation

Our programmable PKC coprocessor supports a broad range of operations. To keep complexity under control, we introduced many levels of hierarchy. This section describes the implementation following a top-down approach.

A. The architecture

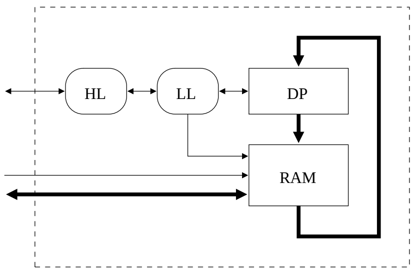

Fig. 1. Architecture of the PKC coprocessor (DP = data path, RAM = data memory, LL = lower-level FSMs, HL = higher-level FSMs, thick lines = data, thin lines = control).

Figure 1 shows the general architecture of our coprocessor, where the thick and the thin lines are used to represent the data flow and the control flow, respectively. The logic controlling the data path (DP) and the data memory (RAM) is divided into two parts, which both consist of Finite State Machines (FSMs):

- the lower-level FSMs (LL): These FSMs are directly controlling the data path and the data memory.
- the higher-level FSMs (HL): To reduce the number of multiplexors, these FSMs are not connected to the data path or the data memory. The HL FSMs control the LL FSMs and take care of the user commands. This keeps complexity under control, because the LL functions are reused for different HL functions.

The data memory is not only accessible by the LL FSMs, but also by the user. Before sending a command to the coprocessor, the user writes the input operands and parameters to predefined addresses in the data memory. After the coprocessor has executed the command, the user reads the result from a predefined address in the data memory.

In the next three paragraphs, we describe the implementation in more detail. Section IV-B elaborates on the data path, the data memory and the interaction with the lower-level FSMs, while Sect. IV-C explains how the higher-level FSMs work. Section IV-D addresses the countermeasures that are implemented to prevent DPA attacks.

B. Controlling the Data Path

In Fig. 2 the data path, data memory and lower-level FSMs are depicted in more detail. Based on this figure, these three parts are described in the following paragraphs.

1) Data Path: The data path consists of three main parts:

- a $\mathrm{GF}(p)$ arithmetic unit: This unit performs an integer multiplication using the dedicated multipliers on the FPGA. Furthermore, it contains and an integer adder/subtracter.
- a $\mathrm{GF}(2^n)$ arithmetic unit: This unit consists of a $\mathrm{GF}(2^n)$ Montgomery multiplier and a $\mathrm{GF}(2^n)$ adder.
- two shift registers for the key (denoted by $<<$ and $>>$ in Fig. 2): Because our coprocessor evaluates an RSA key from MSB to LSB and an ECC key LSB to MSB, a left-shift and a right-shift register are foreseen in the data path. The reason for this difference in key evaluation comes from the fact that ECC uses smaller operand lengths than RSA. This makes it possible to perform an elliptic curve point doubling and addition in parallel without increasing the size of the data path. Point multiplication algorithms evaluating the key from LSB to MSB inherently allow this kind of parallelism [14]. For RSA, the bigger operand size made us choose for a more memory-efficient MSB to LSB algorithm [14].

2) Data Memory: For the data memory, the dedicated RAM blocks on the FPGA are used in dual-port configuration. One port is controlled by the user, the shift registers and the $\mathrm{GF}(p)$ unit, while the other one is controlled by the $\mathrm{GF}(p)$ unit.

3) Lower-level FSMs: The lower-level control part consists of five FSMs:

- $integer\ mul$: FSM controlling the integer multiplier.
- $integer\ add/sub$: FSM controlling the integer adder/subtracter
- $\mathrm{GF}(2^n)\ mul$: FSM controlling the $\mathrm{GF}(2^n)$ multiplier
- $\mathrm{GF}(2^n)\ add$: FSM controlling the $\mathrm{GF}(2^n)$ adder
- key: FSM controlling the shift registers for the evaluation of the key bits

The first four FSMs execute the same general sequence. First, they transfer the data from the data memory into the input registers of one of the arithmetic units. Next, they control the operation to be performed. Finally, they make sure the result is written into the required address in the data memory.

The key FSM addresses the data memory to load the key into one of the key registers and controls the shift operation when the next bit needs to be evaluated.

C. Executing the User Commands

The higher-level FSMs are horizontally divided into seven groups according to their operand length. Figure 3 shows these groups, which are all connected to the user through a control bus. They all receive the same 8-bit user command, but only one group recognizes the command value and executes it.

The “pass” group consists of only one FSM, which makes sure the user can reach the lower-level functions. The other groups consist of many levels of hierarchy in the vertical

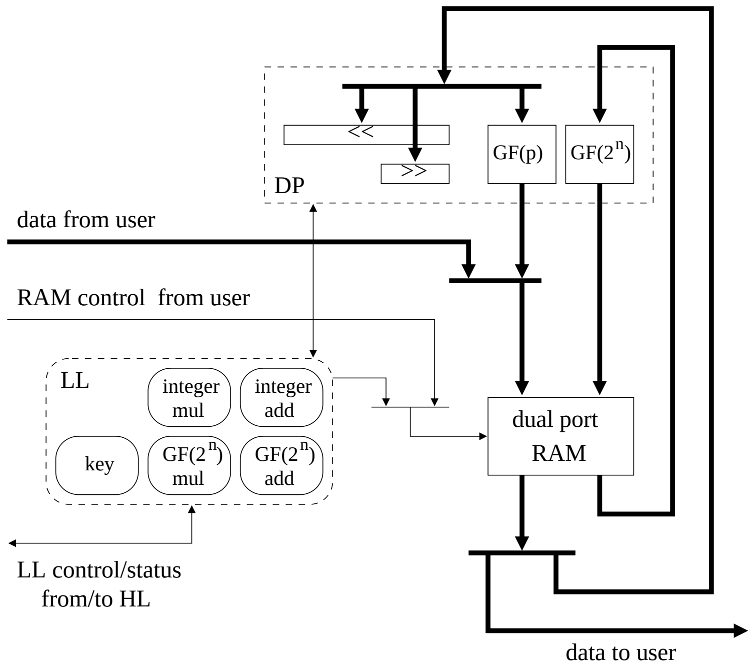

Fig. 2. Data path, RAM and lower-level FSMs of the PKC coprocessor.

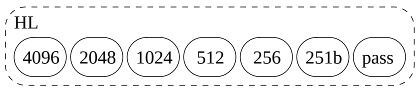

Fig. 3. Higher-level FSMs of the PKC coprocessor.

direction. As an example, these levels are depicted in Fig. 4 for group 512 and 256.

To explain our methodology, we elaborate on the left side of Fig. 4, which shows the hierarchy of operations for group 512. The top level function or level 1 function in group 512, `CRT_w`, performs RSA-CRT with key blinding as a countermeasure for DPA attacks. `CRT_w` uses the level 2 function `RSA_w`, *i.e.* modular exponentiation with key blinding, and the level 5 functions `submod` and `mulmod`, *i.e.* modular subtraction and modular multiplication, respectively. Furthermore `CRT_w` needs an integer multiplication and addition, which is the reason for its connection with the LL FSMs. The key blinding in `RSA_w` is executed through the LL FSMs, while the level 3 function `RSA_wo` performs a modular exponentiation, consisting of only modular multiplications (level 5). The level 5 functions are only connected to the LL functions.

This methodology has been applied in a systematic way to the other operations. The result is a coprocessor which supports RSA up to 4096 bits, RSA using CRT up to 8192 bits, ECC in $\mathrm{GF}(p)$ up to 256 bits and ECC in $\mathrm{GF}(2^n)$ up to 251 bits.

To provide a flexible solution, we made all the levels in the hierarchy accessible. In this way, unimplemented or new cryptographic algorithms can still be executed by the user, which makes our PKC coprocessor fully programmable.

Note that modular multiplication in $\mathrm{GF}(p)$ is implemented as a higher-level function using the integer multiplication and addition that are provided by the data path, while modular multiplication in $\mathrm{GF}(2^n)$ is implemented directly in the data path. The reason for this difference is that the FPGA provides dedicated integers multipliers, which can only be used for $\mathrm{GF}(p)$ multiplication. The $\mathrm{GF}(2^n)$ multiplier is implemented completely on the reconfigurable slices of the FPGA, which makes it more advantageous to provide a modular multiplier directly controllable by the lower-level FSMs.

D. *Side-channel Security*

Because FPGA implementations have been repeatedly shown to be side-channel susceptible (see [17], [22], [24]), we implemented several countermeasures against side-channel attacks. This can be seen from Fig 4, which also shows that some operations can be performed with or without countermeasures, allowing the user to decide on the trade-off between security and performance. For completeness, we briefly summarize the implemented countermeasures in this section.

1) Time-constant Operations: To prevent SPA attacks, we made sure that all operations are time-constant.

For RSA, we implemented the k-ary method to speed up modular exponentiation [14]. When multiplication is not skipped, this method has no conditional branches. To make the power graphs of the squarings and the multiplications look similar, they are performed using the same multiplier.

For ECC, we made sure that the algorithms for point doubling and point addition are balanced, *i.e.* in every step of the algorithms, the same finite field operations are performed.

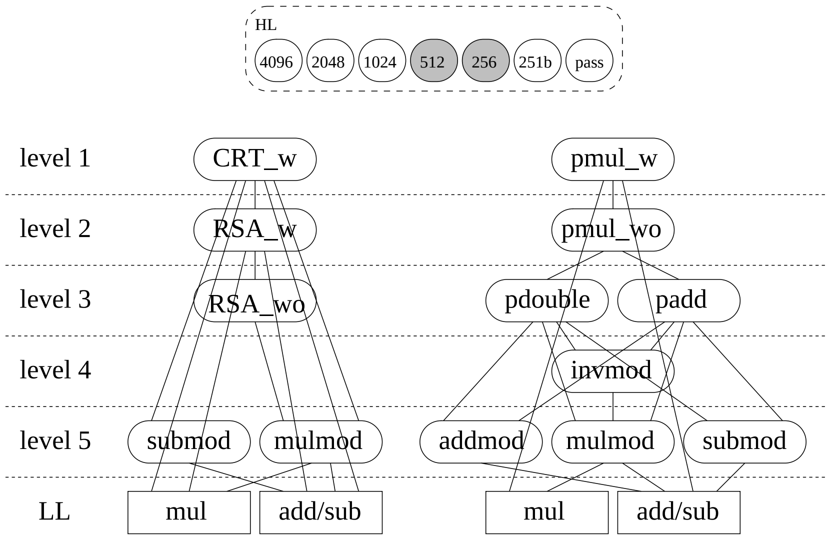

Fig. 4. Hierarchy for the 512 group (left) and the 256 group (right).

TABLE I
IMPLEMENTATION RESULTS FOR OUR PKC COPROCESSOR.

operating frequency	66 MHz
# FPGA CLBs	4826 (58% of the FPGA)
block RAM	66 x 16 kbit
dedicated multipliers	66 x (16x16)

Just like for RSA, finite field squarings and multiplications are executed on the same multiplier.

2) Key Blinding: To prevent first-order DPA attacks, we implemented key blinding for both RSA and ECC, which was explained in Sect. II-C. The examples in Fig. 4 clearly show how this is reflected in the hierarchy of instructions: the top level functions use the LL FSMs for integer multiplication and integer addition.

3) Point Randomization: Elliptic curve point doubling and addition are performed using projective coordinates. As explained in Sect. II-C, this allows for point randomization to be included. This countermeasure is added to the algorithms for point doubling and addition.

V. IMPLEMENTATION RESULTS AND COMPARISON

Table 1 gives the implementation results of our PKC coprocessor, which was implemented on an XC3S5000-5FG1156. The synthesis and place & route were done using Xilinx ISE 8.1, with small area as optimization goal.

The reason that we did not try to gain performance by filling up the FPGA more than 58%, is that we wanted a compact version for embedded applications. Because many of these applications do not have a high speed clock, it does not make sense to push the implementation for speed. This way, we also leave room for other functionality to be added later.

Table 2 compares our implementation to other programmable coprocessors on FPGA.

It is hard to make a fair comparison, because different implementation platforms were used. Moreover, our solution includes RSA as well as ECC, where ECC is supported for both prime and binary extension fields. The other implementations only support a subset of these operations. However, we can conclude that our implementation shows competitive results for RSA compared to previously designed coprocessors on more advanced FPGA platforms. This was achieved by optimally utilizing the dedicated features on the FPGA, such as RAM blocks and multipliers.

The reason that the other implementations in the table outperform our solution for ECC, is that we added security on many levels of the design, as explained in Sect. IV-D. Moreover, our implementation gives a general solution without fixing polynomials or primes and shows results for larger primes. Also, combining ECC with RSA deteriorates the performance of ECC, because a lot of resources are used to support RSA, which makes the area-speed trade-off disadvantageous for the latency of ECC.

VI. CONCLUSIONS AND FUTURE WORK

We presented a programmable coprocessor that provides both RSA and ECC, where ECC is supported in prime as well as binary extension fields. To provide complete flexibility, we made all underlying operations accessible for the user by adopting a hierarchical approach. Complexity is kept under control by modularity: higher-level functions are sharing the lowest-level operations. The result is an FPGA implementation that is optimized for compactness and therefore leaves enough room for other functionality to be added in the future.

ACKNOWLEDGMENT

Nele Mentens, Kazuo Sakiyama and Lejla Batina are partially funded by FWO (G.0450.04), IBBT, EU IST FP6 projects ECRYPT and SESOC.

TABLE II
COMPARISON OF OUR SOLUTION TO OTHER PROGRAMMABLE COPROCESSORS ON FPGA.

reference	implementation platform	comments
this work	XC3S5000	all operations were implemented fully programmable solution countermeasures included
[3]	XC40250XV	only RSA-CRT was implemented up to 1024 bits
[13]	XC2V6000	only RSA-CRT was implemented up to 1024 bits
[16]	XCV400E	only ECC in GF(2^n) up to 255 bits

reference	latency			frequency (MHz)	area
	RSA1024 CRT	ECC GF(p)	ECC GF(2^n)		
this work	4.0 ms	26.8 ms	21.8 ms in GF(2^{251})	66	4826 CLBs 66 RAM blocks 66 mults
[3]	3.1 ms			63.7	6826 CLBs
[13]	2.6 ms			100.5	24767 slices
[16]			0.21 ms in GF(2^{163})	76.7	3002 LUTs 1769 FFs 10 RAM blocks

REFERENCES

[1] L. Batina, S.B. Örs, B. Preneel, and J. Vandewalle. Hardware Architectures for Public Key Cryptography. *Elsevier Science Integration the VLSI Journal*, 34(1-2):1–64, 2003.

[2] M. Bednara, M. Daldrup, J. Teich, J. von zur Gathen, and J. Shokrollahi. Tradeoff Analysis of FPGA Based Elliptic Curve Cryptosystems. In *Proceedings of The IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 797–800, Scottsdale, Arizona, USA, May 26-29 2002.

[3] T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers*, 50(7):759–764, 2001.

[4] Xilinx: The Programmable Logic Company. http://www.xilinx.com, 2006.

[5] J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Ç. K. Koç and C. Paar, editors, *Proceedings of 1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 1717 in Lecture Notes in Computer Science, pages 292–302, Worcester, Massachusetts, USA, August 12-13 1999. Springer-Verlag.

[6] J. Goodman and A.P. Chandrakasan. An energy-efficient reconfigurable public-key cryptography processor. *IEEE Journal of Solid-State Circuits*, 36(11):1808–1820, November 2001.

[7] J. Grosschädl. The Chinese remainder theorem and its application in a high-speed RSA crypto chip. In *Proceedings of the 16th Annual Computer Security Application Conference*, pages 384–393, New Orleans, Louisiana, USA, December 11-15 2000. IEEE Computer Society Press.

[8] IEEE P1363. Standard specifications for public key cryptography, 1999.

[9] N. Koblitz. Elliptic curve cryptosystem. *Math. Comp.*, 48:203–209, 1987.

[10] N. Koblitz. *A Course in Number Theory and Cryptography*, volume 114 of *Graduate text in mathematics*. Springer-Verlag, Berlin, Germany, second edition, 1994.

[11] P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks. http://www.cryptography.com/dpa/technical, 1998.

[12] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology: Proceedings of CRYPTO*, number 1666 in Lecture Notes in Computer Science, pages 388–397, Santa Barbara, CA, USA, August 15-19 1999. Springer-Verlag.

[13] C. McIvor, M. McLoone, J. McCanny, A. Daly, and W. Marnane. Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures. In *Proceedings of 37th Annual Asilomar Conference on Signals, Systems and Computers*, pages 379–384, November 2003.

[14] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[15] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology: Proceedings of CRYPTO*, number 218 in Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1985.

[16] G. Orlando and C. Paar. A high-performance reconfigurable elliptic curve processor for GF(2^m). In Ç.K. Koç and C. Paar, editors, *Proceedings of 2nd International Workshop on Cryptograpic Hardware and Embedded Systems (CHES)*, number 1965 in Lecture Notes in Computer Science, pages 41–56. Springer-Verlag, 2000.

[17] S. B. Örs, E. Oswald, and B. Preneel. Power-analysis attacks on an FPGA – first experimental results. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 2779 in Lecture Notes in Computer Science, pages 35–50, Cologne, Germany, September 7-10 2003. Springer-Verlag.

[18] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[19] A. Satoh and K. Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers, special issue on cryptographic hardware and embedded systems*, 52(4):449–460, April 2003.

[20] P. Schaumont and I. Verbauwhede. A reconfiguration hierarchy for elliptic curve cryptography. In *Proceedings of 35th Asilomar Conference on Signals, Systems, and Computers*, November 4-7 2001.

[21] P. Schaumont, I. Verbauwhede, K. Keutzer, and M. Sarrafzadeh. A quick safari through the reconfiguration jungle. In *Proceedings of 38th Design Automation Conference (DAC'01)*, pages 172–177, Las Vegas, NV, USA, June 18-22 2001.

[22] F.-X. Standaert, S. B. Örs, and B. Preneel. Power Analysis of an FPGA: Implementation of Rijndael: Is Pipelining a DPA Countermeasure? In M. Joye and J.-J. Quisquater, editors, *Proceedings of 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 3156 in Lecture Notes in Computer Science, page 3044. Springer-Verlag, 2004.

[23] J. Wolkerstorfer. Dual-field arithmetic unit for $GF(p)$ and $GF(2^m)$. In B.S. Kaliski Jr., Ç.Koç, and C. Paar, editors, *Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 2523 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[24] T. Wollinger and C. Paar. *Security aspects of FPGAs in cryptographic applications*, chapter in New Algorithms, Architectures, and Applications for Reconfigurable Computing. Kluwer, 2004.