Parallel FPGA Implementation of RSA with Residue Number Systems

— Can side-channel threats be avoided? —

Mathieu Ciet Michael Neve Eric Peeters Jean-Jacques Quisquater

***Abstract*— In this paper, we present a new parallel architecture to avoid side-channel analysis such as: Timing attack, simple/differential power analysis, fault induction attack and simple/differential electromagnetic analysis. We use a Montgomery Multiplication based on Residue Number Systems. Thanks to RNS, we develop a design able to perform an RSA in parallel on a set of identical and independent coprocessors. Of independent interest, we propose a new DPA countermeasure when RNS are used that is only (slightly) memory consuming. Finally, we synthesized our new architecture on FPGA and it presents promising performances results. Even if our aim is to sketch a secure architecture, the RSA signature is performed in less than 150 ms, with competitive hardware resources. To our knowledge, this is the first proposal of an architecture counteracting electromagnetic analysis apart from hardware countermeasures reducing electromagnetic radiations.**

I. Introduction

IMPLEMENTATION of public key cryptography requests the manipulation of large numbers, typically 1024 bits for most of the current applications like RSA [1]. That is the reason why Residue Number Systems (RNS for short) can be very useful. RNS have the main advantage of fast additions, fast multiplications, carry-free, high speed arithmetic, some fault detection, possible error correction and foremost *parallel implementations*.

Many studies have been carried out on RNS [4], [5], [24], [25], [27], but, as far as we know, only Kawamura *et al.* propose a full hardware implementation of RSA with RNS in [24]. They describe a new RNS base extension algorithm and implement the whole system in a LSI prototype based on the Cox-Rower architecture [18] that lays in an efficient bases conversion.

Another important objective for crypto-algorithms implementation is to counteract side-channel analysis. Physical and side-channel attacks refer to attacks that exploit the system implementation. Avoiding these attacks requires hardware and software countermeasures. We consider here countermeasures that can be directly added in the design of a processor. Kocher *et al.* introduced the notion of *side-channel analysis* in [19], [20] and showed the importance for an implementation to be resistant against side-channel analysis and leakages from power consumption. Resistance against fault analysis [8], [9] is another issue: Sensitive information may leak when the cryptosystem operates under unexpected conditions. More recently, in [13], [23] a new type of analysis has been presented, based on electromagnetic radiations of the processor when a crypto-algorithm is processed, (see also [3]), called *ElectroMagnetic Analysis* (EMA for short).

In this paper, we try to tackle the problem at its root in order to design an architecture that can resist some side-channels attacks. We develop here a design able to perform a RSA in parallel, in a set of identical and independent coprocessors denoted *cells*.

This paper is organized as follows: In Section II, we briefly introduce basics on Residue Number Systems. Then, in Section III, Montgomery multiplication in RNS is presented. Section IV deals with implementation strategies. Generalized Mersenne numbers allow us to reduce memory requirements compared to moduli of form $2^\kappa - 1$, but also efficient modular reduction. Then, we explain what we use to counteract each side-channel analysis: A new differential power analysis countermeasure is presented and a "mobile" architecture to avoid electromagnetic analysis is presented. Further on, in Section V, implementations results are presented. We simulate our new architecture on FPGA and give our performance results. Finally, in Section VI, we conclude.

II. Residue Number Systems

Let us introduce the basic terminology [5]:

i) The vector $\{m_1, m_2, \ldots, m_k\}$ forms a set of moduli, called the RNS-base β, where the m_i's are relatively prime.

ii) M is the product $\prod_{i=1}^{k} m_i$ and defines the dynamic range of the system.

iii) The vector $\{x_1, x_2, \ldots, x_k\}$ is the RNS representation of an integer X, less than M, where $x_i = \langle X \rangle_{m_i} = X \bmod m_i$. Any integer X belonging to $\mathbb{Z}/M\mathbb{Z}$ has a unique representation, in base β.

iv) The operation of addition, subtraction and multiplication are defined over $\mathbb{Z}/M\mathbb{Z}$ as:

$$A \pm B = (\langle a_1 \pm b_1 \rangle_{m_1}, \ldots, \langle a_k \pm b_k \rangle_{m_k}) \quad ,$$

$$A \times B = (\langle a_1 \times b_1 \rangle_{m_1}, \ldots, \langle a_k \times b_k \rangle_{m_k}) \quad .$$

These equations illustrate the parallel carry-free nature of the RNS.

v) The reconstruction of X from its residues $\{x_1, x_2, \ldots, x_k\}$ is based on the Chinese Remain-

M. Ciet, M. Neve, E. Peeters and J.-J. Quisquater are with the UCL Crypto Group, Belgium. http://www.dice.ucl.ac.be/crypto/
Contact author: Eric Peeters (peeters@dice.ucl.ac.be)

der Theorem:

$$X = \left\langle \sum_{i=0}^{k} \langle \alpha_i x_i \rangle_{m_i} M_i \right\rangle_M ,$$

where

$$M = \prod_{i=1}^{k} m_i; \quad M_i = \frac{M}{m_i}; \quad \alpha_i = \langle M_i^{-1} \rangle_{m_i} .$$

vi) The vector $\{x'_1, \ldots, x'_k\}, 0 \leq x'_i < m_i$ is the Mixed Radix System (MRS) representation of an integer X less than M, such that:

$$X = x'_1 + x'_2 m_1 + x'_3 m_1 m_2 + \ldots + x'_k \prod_{i=1}^{k-1} m_i .$$

vii) Comparison and division are very difficult operations to perform on the RNS representation [15], [16], [31]. That is the reason why Montgomery multiplication is well suited to RNS.

III. Montgomery Multiplication in RNS

In 1985, Montgomery introduces a method [22], widely used nowadays, for modular multiplication that requires no division. Let R be an integer such that $\gcd(R, N) = 1$ and $N' = -N^{-1} \mod R$ then the following equation holds:

$$\frac{P + (PN' \mod R)N}{R} \equiv PR^{-1} \mod N . \qquad (1)$$

This method was adapted to RNS by Posch and Posch [25]. Referring to Eq. (1), they propose to pick $\widetilde{M}$ (product of elements of a base) as R. So that, all operations of multiplication and addition can be performed in parallel. However, division requires a base extension. Two RNS bases β and $\widetilde{\beta}$ are chosen large enough to represent all intermediate results.

If $4N < M, \widetilde{M}$ then output length matches input length of the next modular multiplication of an exponentiation.

Let us briefly recall how the Montgomery multiplication algorithm in RNS proceeds [18]:

Input: $\langle A \rangle_{\beta \cup \widetilde{\beta}}, \langle B \rangle_{\beta \cup \widetilde{\beta}}$ (where $A, B < 2N$)
Output: $\langle R \rangle_{\beta \cup \widetilde{\beta}}$ (where $R \equiv AB\widetilde{M}^{-1} \mod N, R < 2N$)

Base β Operation	Base $\widetilde{\beta}$ Operation
$\langle s \rangle_\beta \leftarrow \langle A \rangle_\beta . \langle B \rangle_\beta$	$\langle s \rangle_{\widetilde{\beta}} \leftarrow \langle A \rangle_{\widetilde{\beta}} . \langle B \rangle_{\widetilde{\beta}}$
–	$\langle q \rangle_{\widetilde{\beta}} \leftarrow \langle s \rangle_{\widetilde{\beta}} . \langle -N^{-1} \rangle_{\widetilde{\beta}}$
$\langle q \rangle_{\beta \cup \widetilde{\beta}} \Longleftarrow \langle q \rangle_{\widetilde{\beta}}$	
$\langle r' \rangle_\beta \leftarrow \langle q \rangle_\beta . \langle N \rangle_\beta$	–
$\langle r'' \rangle_\beta \leftarrow \langle s \rangle_\beta + \langle r' \rangle_\beta$	–
$\langle R \rangle_\beta \leftarrow \langle r'' \rangle_\beta . \langle \widetilde{M}^{-1} \rangle_\beta$	–
$\langle R \rangle_\beta \Longrightarrow \langle R \rangle_{\beta \cup \widetilde{\beta}}$	

Fig. 1. Montgomery Multiplication algorithm in RNS

Many different methods (more or less efficient depending on the number of elements in the base) were proposed to perform the base extension step. The most common are: Conversion using MRS [5], conversion using an extra modulus [28], conversion allowing an offset [4], approximate base conversion [25] and error-free approximate base conversion [18].

IV. Hardware Implementations Strategies

In this section, we present the various strategies used to implement RSA on FPGA using RNS.

A. Mersenne numbers

When $N = 2^\kappa - 1$ is prime, N is usually said to be a *Mersenne number* (or a *Mersenne prime*). Throughout the paper, for the sake of simplicity, we extend this definition to any number of the form $2^\kappa - 1$, be it prime or not. Its particular form allows modular reduction $A \mod N$ with at most two κ-bit additions, for any $A < N^2$. This shortcut provides a major speed-up in algorithms based on modular arithmetic.

Base elements uniquely composed of Mersenne numbers are in a large range since they must be relatively prime. This leads to inefficient hardware implementations since every operation should be thought for the longest element. For example, considering a suiting base for 1024-bit representations: $\beta = \{2^{\kappa_1} - 1, \ldots, 2^{\kappa_{13}} - 1\}$ with $\kappa_i = 37, 47, 59, 67, 73, 83, 97, 103, 109, 119, 123, 125$ and similarly for $\tilde{\beta}$. To be timing analysis resistant, the implementation must deal with all data as 124-bit element.

Pairwise elements ($2^{\kappa_1} \pm 1$) as suggested in [29] could lead to better balanced bases. However, it does not completely fix this issue and more seriously adds higher design complexity.

Generalized Mersenne numbers [30], [34], *i.e.* $2^{\kappa_1} - 2^{\kappa_2} - 1$, on the other hand, permit bases in smaller range. For $N = 2^{\kappa_1} - 2^{\kappa_2} - 1$ and $A < N^2$, the modular reduction $A \mod N$ takes at most 6 additions of κ_1-bits, if $0 < \kappa_2 < \frac{\kappa_1 + 1}{2}$. The careful reader sees that the remainder of the reduction might require a final substraction of N to be completely reduced. This issue happens to be only required during the base conversions where the moduli are mixed, as in MRS conversion.

B. Side-channels countermeasures

The most generally known side-channel analysis is *Timing Attack* presented by Kocher [19], see also [26] in the case of use of CRT. The countermeasures consist of a modification of the well-known Montgomery multiplication [12], [14], [32], [33], *i.e.* avoiding the final substraction such as obtaining timing independent processes.

There is another side-channel analysis that uses power consumption, suggested by Kocher *et al.* in [20]. Two families have to be considered. The first one uses a simple trace of a power consumption and is called *Simple Power Analysis* (SPA for short). The second one is more sophisticated and needs several power traces, see [2], [6], [10]. This is called *Differential Power Analysis* (DPA for short). To avoid SPA we use a variant of the 'square-and-multiply always' algorithm. Randomization of the message and of the exponent are classically used to defeat DPA. However, since RNS is used, we can *independently* randomize each base. We also propose another method (that can be combined with the most classical ones) and that is based on the fact that we use special numbers for RNS. Indeed, since we use relatively primes of form $2^{\kappa_1} - 2^{\kappa_2} - 1$, we are able to store several of these bases with a small memory requirement and randomly select the bases for each process. Indeed, the use

of generalized Mersenne numbers (prime to each others), as already seen, has many nice properties. If these bases are such that $58 \leq \kappa_1 \leq 64$ and $0 < \kappa_2 < \frac{\kappa_1+1}{2}$, we are able to generate at least 32 bases with very small memory requirements: κ_1 is represented as its distance to 64, and κ_1 to 0. Only 9 moduli are needed, then there are more than 2^{24} possible combinations. Then, each base is randomly chosen for each RNS exponentiation, in combination with the more classical randomization methods of message and exponent *for each base*. Just remark that our countermeasure is only (small) memory consuming[1]. It is also worth noticing that it is very simple to desynchronize each cell's process by adding random delays.

Another threat must be taken into account. It is called *Fault Induction Attacks* (also sometimes 'Differential Fault Analysis', DFA for short) [8], [9]. We decided to combine two countermeasures. The first one has been proposed by Yen *et al.* in [35]. To our knowledge, this is the best way to prevent fault attack against DFA, since no "if" test is needed. The second one is directly related to the use of RNS. Single-error detection can be done using redundant modulus m_r such that: $\forall i \in \{1 \cdots k\}, m_i < m_r$. The error is detected by checking if the converted value is outside the range $[0, M-1]$, see [31] for further details.

Finally, recently a new type of attack has been proposed, based on electromagnetic radiations of the chip on which the crypto-algorithm is processed [13], [23]. This attack is called *ElectroMagnetic Analysis* (EMA for short), with its counterpart using a statistical treatment *Differential Electro-Magnetic Analysis* (DEMA for short). It seems difficult to provide software countermeasures to counteract this type of attack. It is not really clear either how to proceed in hardware apart from the reduction of electromagnetic emanations. One of our aims is to design a new type of architecture that makes it harder for an attacker to mount an EMA/DEMA. The basic idea is to design an architecture with independent cells that randomly proceed. Indeed, one of the advantage of EMA in comparison of PA is that better *Signal-to-Noise Ratio* (SNR) is obtained, since the probe can be located above the interesting area with much less taking into account the emanation produced by the rest of the chip.

Let μ be the hashed and padded message to be signed. The secret exponent d is given in its CRT form: $d_p = d \bmod p - 1$, $d_q = d \bmod q - 1$. The principle of our architecture is as follows:

i) Compute $\mu^{(p)} = \mu \bmod p$ and $\mu^{(q)} = \mu \bmod q$,
ii) For each $\mu^{(i)}$, randomly choose a base β: $\{m_1, \cdots, m_k\}$,
iii) Represent each $\mu^{(i)}$ in this base as $\{\mu_1^{(i)}, \cdots, \mu_k^{(i)}\}$, where $\mu_j^{(i)} = \mu^{(i)} \bmod m_j$,
iv) Add randomizations to avoid DPA to each $\mu_j^{(i)}$,
v) Compute the exponentiation using RNS-Montgomery multiplication for each $\mu_j^{(i)}$ with $d_i^{(j)}$; at each operation randomly choose a cell,
vi) Use the CRT to recover $\left(\mu^{(i)}\right)^{d_i} \bmod i$,
vii) Use the CRT and output $\mu^d \bmod N$.

More details of our architecture are given in Section V.

[1]This is the first DPA countermeasure for RSA which is for free in complexity.

Summarizing our strategies implementations, we can simply say that we have taken into account all side-channel constraints at the beginning of the implementation design in such a way that attacks are defeated or at least really more difficult to exploit.

V. Implementation Results

A. Implementation

The main core of our design is a set of all alike, independent and parallel coprocessors. They can perform basic modular operations using any modulus of the bases. Each cell is connected to one common 16-bit wide communication bus on which they can interact with a defined protocol. They are managed by a *multiplier controller* containing the sequence of operations to perform one 512-bit multiplication of the 'square-and-multiply' algorithm. The combination of the control unit and the set of cells forms a large *multiplication processor*. Following the current key bit, the *exponentiation processor* feeds (A, A) or (A, X) into the multiplication processor. However, the latter processes the given data in the same way, no matter if the operation is a square or a multiply. Fig. 2 illustrates all parts connected together by the *data* bus.

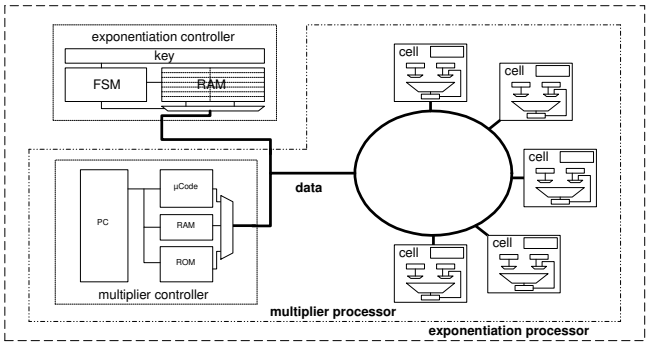

Fig. 2. Architectural structure of the 512-bit exponentiation processor

We implemented the cells by using the less hardware resources as possible with minimal loss in frequency. The multiplication seems the critical operation in terms of resources/time tradeoff, then we proposed different strategies to handle it. All models were described using VHDL.

Our first try was based on *Scaling Accumulator* technic. So the principle is to compute a complete multiplication, to store the results in registers, and finally to process the modular reduction by carrying out the different numbers to be added (see [34]). As explained only 4 additions are required. Based on this structure, modular addition and subtraction can also be easily treated.

Another direction is the following. Instead of separating the operation and the reduction, it seems interesting to interleave both. At each step, the result (say S_1) from addition of previous outcome and new partial product is truncated after κ_1 bit (S_1'). The overflowing bits (MSBs of S_1) are then used to produce the remainder of the number (say R_1). Finally S_1' is added to R_1. If $\kappa_2 \geq 4$ is verified, the remainder generation is simplified. Using some shifters, it has been easily adapted to a pipelined version. Implementation details of this cell are given in Fig. 3.

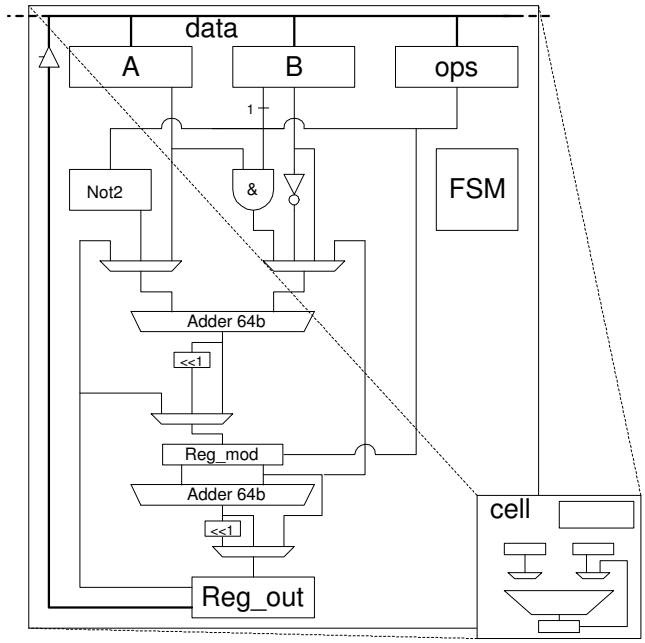

Fig. 3. Detail of a 2-stage pipeline cell

As explained in § IV-A, the remainder might not be fully reduced and it could lead to a problem, especially in bases conversion. An extra final substraction is performed to tackle this issue. If an underflow occurs, the previous value is kept and the other one is discarded. This extra step protects against conversion error in a simple and timing-resistant way.

The purpose of the multiplication processor is to orchestrate the set of cells in order to perform one 512-bit multiplication. The operands expressed in the 9 moduli of both bases are sent through *data* by the exponentiation processor. Once the data are set, the multiplication processor shares out the computations to idle cells. When the result of a cell is available, the corresponding cell uploads its result. And so on till the whole multiplication is completed. The global results in both bases are sent back to the exponentiation processor. There is clearly no difference in the operation sequence whether the general computation is a square or a multiply.

It is basically built as a microcontroller: the 10-bit program counter refers to the current operation and to the address of the operands in the memory; the microcode is fetched and then interpreted by the processor which enables the memory to send or receive data in burst mode.

The 512 bits of the secret key is stored in a register inside the exponentiation processor. The latter is scanned from left to right to perform a variant of the 'square-and-multiply-always' algorithm.

B. Performances

The average number of clock cycles per operation (addition, subtraction, multiplication or write back) is about 11. Hence, if the operations are well scheduled in the program, a basic operation in the base β or $\tilde{\beta}$ requires virtually 22 cycles (a multiplication requires a bit more than 64 cycles).

We choose to implement the RNS Montgomery multiplication algorithm proposed by Bajard *et al.* [4]. Indeed, in [17], the authors claim that their algorithm is more efficient than Kawamura's one. Their algorithm uses two different base conversions allowing to use less basic operations than other algorithms. However, they specified a more restrictive constraint over the choice of the base β and $\tilde{\beta}$: $(k+2)^2N < M, \tilde{M}$. This constraint could increase the number of element required for one base but 9 moduli are still convenient in our case. With this algorithm, a Montgomery multiplication requires $2k^2+10k+4$ multiplications, $2k^2+k-1$ additions and $k+1$ subtractions. So a total of $4k^2+12k+4$ operations (436 operations in case of $k=9$).

We synthesized our design on a Virtex2 xc2v6000 clocked at 50 Mhz. The synthesis is performed with FPGA Compiler 2 3.7.1 (SYNOPSYS) and the implementation with XILINX ISE-5. Moreover, we consume 9 RAM blocks, 4956 slices whose them 2788 registers and 8185 LUTs, for a 512 bits exponentiation. The signature is achieved in 147 ms. Since the goal is to evaluate our methodology on general reconfigurable hardware devices, we did not take the most of particular features of Virtex2 (such as internal 18 by 18 bits multipliers). Performing a 1024 bits exponentiation in the same time would only require doubling hardware resources. The design contains the hardware needed to receive the message μ in RNS and perform the exponentiation. Compare our design with a Cox-Rower architecture seems to be very difficult. Indeed, many choices differ from their implementation, especially all the side-channels considerations. But, we could imagine that implementing their architecture in a FPGA is certainly more hardware consuming than ours regarding the number of adders and multipliers they need for one single cell.

VI. CONCLUSION

We have presented a new architecture to avoid side-channels analysis based on timing, power consumptions and electromagnetic radiations that leak when a crypto-algorithm is processed. Countermeasures against fault inductions have also been implemented. Our design is based on Residue Number Systems that permit fast additions and multiplications, carry-free, high speed arithmetic, some fault detection, possible error correction and mostly parallel implementations. We showed that generalized Mersenne numbers provide well balanced bases and offer efficient modular reductions. They can also be used to obtain a new DPA countermeasure that is only memory consuming but that is for free in terms of performance. Moreover, we have proposed different hardware strategies to manage a multiplication within the cells.

Finally, we have given a concrete implementation of our architecture on FPGA. Despite our aim is to design a side-channel secure implementation, our FPGA implementation results in promising efficiency: Less than 150 ms for a 1024-bit RSA, with competitive hardware resources. Our implementation cannot be compared with those proposed by Nozaki *et al.* since objectives are really different, and different circuit types are used, and less hardware requirements are needed in our case.

REFERENCES

[1] IEEE Std 1363-2000. *IEEE Standard Specifications for Public-Key Cryptography*. IEEE Computer Society, August 29, 2000.

[2] Mehdi-Laurent Akkar, Régis Bevan, Paul Dischamp, Didier Moyart. Power analysis, What is now possible.... In T. Okamoto, Ed., *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pp. 489–502. Springer-Verlag, 2000.

[3] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao and Pankaj Rohatgi The EM side-channel(s). In B.S. Kaliski Jr. and Ç.K. Koç, Ed., *Cryptographic Hardware and Embedded Systems (CHES 2002)*, volume 2523 of *Lecture Notes in Computer Science*, pp. 29–45. Springer, 2002.

[4] Jean-Claude Bajard, Laurent-Stéphane Didier and Peter Kornerup, Modular Multiplication and Base Extensions in Residue Number Systems, *IEEE Symposium on Computer Arithmetic*(2001).

[5] Jean-Claude Bajard, Laurent-Stéphane Didier and Peter Kornerup, *An RNS Montgomery modular multiplication algorithm*, IEEE Transactions on Computers, vol. 47, pp. 766-76, July 1998.

[6] Régis Bevan and Erik Knudsen. Ways to enhance differential power analysis. In P.J. Lee and C.H. Lim, Ed., *Information Security and Cryptology (ICISC 2002)*, volume 2587 of *Lecture Notes in Computer Science*, pp. 327–342. Springer-Verlag, 2002.

[7] Dan R. Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, Ed., *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pp. 37–51. Springer, 1997.

[8] Dan R. Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14(2):101–119, 2001. An earlier version appears in EUROCRYPT '97 [7].

[9] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In B.S. Kaliski Jr., Ed., *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pp. 513–525. Springer, 1997.

[10] Eric Brier, Christophe Clavier, and Francis Olivier. Optimal statistical power analysis. Cryptology ePrint Archive, Report 2003/152, 2003. `http://eprint.iacr.org/2003/152`.

[11] Jaewook Chung and Anwar Hasan. More generalized mersenne numbers. *Selected Areas in Cryptography (SAC 2003)*, August 14 & 15, 2003, Carleton University, Ottawa, Ontario, Canada.

[12] Jean-François Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards*. PhD thesis, Université catholique de Louvain, May 1998.

[13] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Ç.K. Koç, D. Naccache, and C. Paar, Ed., *Cryptographic Hardware and Embedded Systems (CHES 2001)*, volume 2162 of *Lecture Notes in Computer Science*, pp. 251–261. Springer, 2001.

[14] Gael Hachez and Jean-Jacques Quisquater. Montgomery exponentiation with no final subtractions: Improved results. In Ç.K. Koç and C. Paar, Ed., *Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pp. 293–301, 2000.

[15] Markus A. Hitz and Erich Kaltofen. Integer division in residue number systems. *IEEE Transaction on Computers* 44(8), pp. 983-989,(1995).

[16] Ching Yu Hung and Behrooz Parhami. Fast RNS division algorithms for fixed divisors with application to RSA encryption *Information Processing Letters*, Vol.51, pp. 163-169, 1994.

[17] Laurent Imbert, Jean-Claude Bajard. A full RNS implementation of RSA Research Report 02068, LIRMM, available at: `http://www.lirmm.fr/bibli/GEIDEFile.PDF?Archive=191917991919&File=RR\%2D02068_PDF`, May 2002.

[18] Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo, Cox-Rower Architecture for Fast Parallel Montgomery Multiplication, In B. Preneel Ed. *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000, pp. 523-538.

[19] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, Ed., *Advances in Cryptology - CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pp. 104–113. Springer, 1996.

[20] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In M. Wiener, Ed., *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pp. 388–397. Springer, 1999.

[21] Olivier Kömmerling and Markus G. Kuhn, Design principles for tamper-resistant smartcard processors, *USENIX Workshop on Smartcard Technology (Smarcard'99)*, pp. 9–20. USENIX Association, 1999

[22] Peter L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, 44(170):519–521, April 1985.

[23] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In I. Attali and T.P. Jensen, Ed., *Smart Card Programming and Security (Esmart 2001)*, volume 2140 of *Lecture Notes in Computer Science*, pp. 200–210. Springer, 2001.

[24] Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shinichi Kawamura, Implementation of RSA algorithm based on RNS Montgomery multiplication, In C. Paar ed. *Cryptographic Hardware and Embedded Systems - CHES 2001*, pp. 364376, Springer-Verlag, Berlin, Germany.

[25] Karl C. Posh and Reinhard Posh. Modulo reduction in Residue Number Systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 5, May 1995, pp. 449-454.

[26] Werner Schindler. A timing attack against RSA with the chinese remainder theorem. In Ç.K. Koç and C. Paar, Ed., *Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 2000.

[27] J. Schwemmlein, Karl C. Posch, Reinhard Posch. RNS-modulo reduction upon a restricted base value set and its applicability to RSA cryptography. *Computer & Security*, Vol.17, No.7, pp. 637-650, 1998

[28] A.P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computers*, 38 (1989), pp. 292-297.

[29] Alex Skavantzos and Mohammad Abdallah. Implementation issues of the two-level residue number system with pairs of conjugate moduli IEEE Transactions on Signal Processing, vol. 47, no. 3, pp. 826–838, 1999.

[30] Jérôme A. Solinas Generalized Mersenne numbers. Technical report, The centre for applied cryptographic research, University of Waterloo, 1999. CORR 99-39.

[31] Nicholas S. Szabó an Richard I. Tanaka. Residue arithemtic and its application to computer technology. McGraw-Hill,1967

[32] Colin D. Walter. An overview of Montgomery multiplication technique: How to make it smaller and faster. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES '99)*, volume 1717 of *Lecture Note in Computer Science*, pp. 80–93 1999.

[33] Colin D. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35(21):1831–1832, October 1999.

[34] Huapeng Wu. On modular reduction Technical report, CACR, University of Waterloo, 2000. CORR 2000-36.

[35] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Seongan Moon. RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. In K. Kim, Ed., *Information Security and Cryptology - ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pp. 397–413. Springer-Verlag, 2001.