# Fault and Simple Power Attack Resistant RSA using Montgomery Modular Multiplication

Apostolos P. Fournaris
Hitachi Europe SAS
Information and Communication Technologies Lab. (ICTL)
European R-D Center (ERD), Sophia Antipolis, France
Email: apofour@ieee.com

***Abstract*— Side channel attacks and more specifically fault, simple power attacks, constitute a pragmatic, potent mean of braking a cryptographic algorithm like RSA. For this reason, many researchers have proposed modifications on the arithmetic operation functions required for RSA in order to thwart those attacks. However, these modifications are applied on theoretic - algorithmic level and do not necessary result in high performance RSA designs. This paper constitute the first complete attempt for an efficient design approach on a fault and simple power attack resistant RSA based on the well known, for its high performance, Montgomery multiplication algorithm. To achieve this, a fault and simple power attack resistant modular exponentiation algorithm is proposed that is based on the Montgomery modular multiplication. In order to optimize this algorithm's performance we also propose a modified version of Montgomery modular multiplication algorithm that employs value precomputation and carry save logic in all input, output and intermediate values. We introduce a hardware architecture based on the proposed Montgomery modular multiplication algorithm and use it as a building block for the design of a fault and simple power attack resistant modular exponentiation unit. This unit is optimized by taking advantage of the inherit parallelism in the proposed fault and simple power attack resistant modular exponentiation algorithm. Realizing the proposed unit in FPGA technology very advantageous results are found when compared against other well known designs even though our design bears an extra computation cost due to its fault and simple power attack resistance characteristic.**

## I. Introduction

RSA is one of the most popular public key cryptographic algorithms. It is widely used in public signatures applications, authentication schemes and electronic transactions in general. The mathematics behind this algorithm, are summarized in two operations, modular multiplication ($X \cdot Y mod N$) and modular exponentiation. In the RSA cryptosystem, the arithmetic operation $m^e mod N$ is used, where N is a prime product of two relative prime numbers, $m$ is the message and $e$ the secret key. The modular exponentiation operation is realized as a series of modular multiplication and squaring operations. In order to create an efficient implementation of RSA, one has to design efficiently the multiplication of two modular numbers and utilize this operation in a cost efficient modular exponentiation architecture.

The modular multiplication computation cost, both in hardware resources and computational time, is not negligible and determines the performance of the whole RSA system. Many researchers [1] [2] [3] agree that the use of Montgomery modular multiplication algorithm (MMM), performing the operation $X \cdot Y \cdot R^{-1} mod N$, is the best approach on minimizing the computation cost of modular multiplication regardless of the fact that this algorithm employs one extra constant factor ($R^{-1}$). This constant factor is eliminated in the RSA operations, by redesigning RSA's modular exponentiation algorithm [1].

[0]The work reported in this paper is supported by the European Commission through the SECRICOM FP7 European project under contract FP7 SEC 218123

While RSA is considered very secure against traditional cryptanalysis, side channel attacks (SCA) have been successful in determining RSA keys using information leaking from a straightforward implementation of the algorithm. Fault attacks (FA) and simple power attacks (SPA) are among the most effective SCA launched on RSA. Many researchers have proposed solutions on protecting RSA from FA and SPA with relative success [4], [5], [6], [7]. However, those solutions are focused on an FA-SPA resistance on algorithmic level without taking into account the implementation cost for one FA SPA secure RSA encryption-decryption operation. This cost is associated with the arithmetic operations required for RSA and can be very high - restrictive when applying the existing FA SPA resistant RSA solutions in real security devices.

In this paper, a High performance FA and SPA resistant modular exponentiation unit is proposed. To achieve this, we utilize an optimized version of the MMM algorithm (CSMMM) that employs carry-save logic (C-S logic) in all its inputs, outputs and intermediate values. We also propose an FA SPA resistant modular exponentiation (FSPAMME) algorithm that takes advantage of CSMMM special structure (C-S logic) by completely avoiding transformations from C-S logic values to non C-S logic values in every algorithmic round. Also, an architecture for MMM is proposed, realizing in hardware the CSMMM algorithm. This architecture is used as a building block for a FA and SPA resistant modular exponentiation unit (FASPAMEA) that implements the functionality of the proposed FSPAMME algorithm. Both proposed designs are realized in FPGA technology and measurements are taken in order to verify their efficiency by comparing them with other well known works. The proposed design implementations proved to be very attractive in performance characteristics (area, max. frequency and throughput). Our work constitutes the first attempt to our knowledge, of designing a hardware optimized FA-SPA resistant unit, based on the MMM algorithm for modular multiplication.

The paper is organized as follows. In Section II, MMM and FA SPA resistant modular exponentiation algorithms are presented. In Section III the proposed CSMMM algorithm and architecture are described. In Section IV the FSPAMME algorithm is proposed and analyzed. In section V the proposed FASPAMEA is described. In Section VI, performance measurements, implementation results and comparisons are given and Section VI concludes the paper.

## II. Montgomery Modular Multiplication and Exponentiation

The MMM algorithm [8] calculates the value $A = X \cdot Y \cdot R^{-1} mod N$ where R is a constant number usually $R = 2^n$. The n-bit value N has to be an integer filling the condition gcd(R,N)=1. In RSA, N value is odd as the product of two primes therefore the above constrain is always true for $R = 2^n$. C. Walter in [2] proved that

if the MMM algorithm is to be used for modular exponentiation the final step of the algorithm (a subtraction operation) is not necessary as long as N is odd, $Y < 2N$ and $2N < 2^{p-1}$ where p is the number of algorithmic rounds. So, to satisfy the above constrains it suffices that $p = k + 2$ since the input Y will never be greater than 2N in a modular exponentiation process [2], [9]. Therefore the MMM algorithm has the form described below. We denote the i-th bit of a variable by [i]. Note, that to meet the $p = n + 2$ constrain all the $n$ bit-length numbers have to be extended to $n + 2$ bit length numbers by padding zeros.

***Montgomery Modular Multiplication (MMM) algorithm***

**Input**: $X, Y, N$ **Output**: $A = X \cdot Y \cdot R^{-1} mod N$ **Init**: $A = 0$

1) **For** $k = 0$ **to** $k + 1$
   a) $q = (A[0] + X[k] \cdot Y[0]) mod 2$
   b) $A = (A + X[k] \cdot Y + q \cdot N)/2$
2) **Return** A

### A. Fault Secure Modular Exponentiation

The fault attack (FA) goal is to disturb a hardware device during cryptographic operation execution, analyze the faulty behavior of the disturbed device and as a result deduce sensitive information. Combining such attack with a simple power attack (SPA), where a hardware device's power trace is measured and exploited for secret information leakage [10], a crypto-system attacker can relatively easy deduce a cryptographic key of a secure hardware device. SCA countermeasures can be generic, on circuit level, (more effective against power attacks) [11] [12] or specialized, focused on specific cryptographic algorithms, on an algorithmic level. The second approach can be more effective since it utilizes techniques that better negate a cryptoalgorithm's specialized SCA weaknesses. In RSA, the target of the FA and SPA is the modular exponentiation unit, that constitute the core component in message encryption-decryption. Whether a modular exponentiation unit is used for a straight realization of RSA or through CRT, FA-SPA resistance measures are very specific. SPA resistance is achieved by making the arithmetic operations during the exponentiation algorithm execution undiscriminated from an external observer [6]. This countermeasure can be further enhanced by blinding the modulus N using a random number. Fault attack countermeasures are based on techniques of detecting =the injection of a single fault in a crypto-system and prohibiting the release of secret information. Giraud in [4] proposed a FA- SPA resistant modular exponentiation algorithm and later Kim and Quisquater [7] proposed an attack on the algorithm along with a way to thwart it. The modification of Giraud's RSA algorithm [7] has the following form

***FA-SPA Giraud Modular Exponentiation (FASPAME) algorithm***

**Input**: $m, e = (1, e_{t-2}, ...e_0), N, a$

**Output**: $(a + m^{e-1}) mod N, (a + m^e) mod N$

1) $s_0 = m$
2) $s_1 = m^2 mod N$
3) **For** $i = t - 2$ **to** 1
   a) $s_{\bar{e_i}} = s_{\bar{e_i}} \cdot s_{e_i} mod N$
   b) $s_{e_i} = s_{e_i}^2 mod N$
4) $s_1 = (a + s_1 \cdot s_0) mod N$
5) $s_0 = (a + s_0^2) mod N$
6) **If** (Loop Counter $i$ and exponent $e$ are not modified) **then** return $(s_0, s_1)$ **else** return error

The FASPAME uses Joye et al's algorithm for SPA resistance [6] but is further enhanced by the use of a 32 bit random number $a$ that is added to the results. FA resistance is achieved by introducing two values, $s_o$ and $s_1$, for which the equation $s_1 = m \cdot s_0 mod N$ is always true. If this connection between $s_0$ and $s_1$ is disturbed then a fault attack is detected and the cryptographic processes is canceled. Note that the subscript with a horizontal line over it, refers to the logical not operation of the subscript's value.

## III. Proposed Montgomery Multiplication using Carry-Save logic

In the design of an efficient Montgomery multiplier for use in a modular exponentiation algorithm, the fact that the Montgomery algorithm is expected to be run many times, can not be ignored. Therefore, it is very important that a resulting Montgomery multiplier performs one multiplication very fast and with restricted hardware resources. Several researchers have proposed the use of Carry - Save logic in order to speed up the required additions in each algorithmic round. However, the multiplication result after one Carry - Save addition is in carry save format and can not be used in a following Montgomery multiplication as it is (needs to be transformed from Carry-Save format to regular format). This transformation has a performance cost that limits, considerably, the performance gain from the use of Carry - Save logic. A solution to this problem, proposed by several researchers [13], [14], is rewriting the MMM algorithm so that all its inputs, outputs and intermediate results are in Carry - Save format. This approach along with the use of precomputation gives very promising results, as described in [13]. The MMM algorithm of [13] offers an elegant approach on the problem, however, is far from complete since it does not give a detailed description on the performed calculations. In this paper, a refinement is made on this algorithm and optimizations are proposed as shown below

***Opt. C-S Montgomery Modular Multiplication algorithm (CSMMM(X,Y,N))***

**Input**: $X, Y, N$ **Output**: $C = X \cdot Y \cdot R^{-1} mod N$

*Precomputation*: $YN_S + YN_C = Y_C + Y_S + N$

1) $C_{in}^{(0)} = 0, S_{in}^{(0)} = 0, X_{nextC}^{(-1)} = 0$
2) **For** $k = 0$ **to** $n + 1$
   a) $X_R^{(k)}[k] + X_{nextC}^{(k)} = X_C^{(k-1)}[k] + X_S^{(k-1)}[k] + X_{nextC}^{(k-1)}$
   b) $q^{(k)} = (C_{in}^{(k)}[0] + X_R^{(k)}[k] \cdot Y_S[0]) mod 2$
   c) **If** $(X_R^{(k)}[k] = 0)$ **then**
      i) $I_C^{(k)} = 0$
      ii) **If** $(q^{(k)} = 0)$ **then** $(I_S^{(k)} = 0)$ **else** $(I_S^{(k)} = N)$
      **else**
      i) **If** $(q^{(k)} = 0)$ **then** $(I_S^{(k)} = Y_S\ )$ **else** $(I_S^{(k)} = YN_S)$
      ii) **If** $(q^{(k)} = 0)$ **then** $(I_C^{(k)} = Y_C\ )$ **else** $(I_C^{(k)} = YN_C)$
   d) $C_{in}^{(k+1)} + S_{in}^{(k+1)} = (C_{in}^{(k)} + S_{in}^{(k)} + I_S^{(k)} + I_C^{(k)})/2$
3) **Return** $C_{in}^{(n+2)}, \quad S_{in}^{(n+2)}$

In the CSMMM algorithm, the Y + N value is precomputed in carry-save format before the main loop. Thus there is no need to recalculate this value in each algorithmic round. Also, in CSMMM algorithm a new variable is introduced ($X_R[k]$) that is used for reconstructing the $k - th$ bit of $X$ from its carry-save format ($X_C, X_S$). All the values are in carry save format (even the inputs X and Y). Note that the superscript refers to the current algorithmic round and the brackets [i] refer to the i-th bit of a used value.

## IV. Proposed FA-SPA Modular Exponentiation Algorithm

The FASPAME algorithm seems very promising in terms of security, however, its efficiency in terms of computation speed and utilized hardware resources remains questionable. Modular multiplication utilized in FASPAME has to be implemented efficiently in

hardware so as to result in a antagonistic cryptographic exponentiation design. Most researchers agree that Montgomery modular multiplication is one of the most efficient modular multiplication approaches. However, through this algorithm the performed modular operation differs from the generic form presented in FASPAME ($C = X \cdot Y \cdot R^{-1} mod N$ instead of $C = X \cdot Y mod N$). Therefore, the FASPAME algorithm needs to be modified in order to utilize Montgomery modular multiplication (MMM) as its structural unit. Below, we propose such a modified version of FASPAME that employs the version of MMM (CSMMM) described in section III.

***Proposed FA-SPA Montgomery Modular Exponentiation (FSPAMME) algorithm***

**Input**: $m, e = (1, e_{t-2}, ...e_0), N$

**Output**: $(a + m^{e-1}) mod N, (m^e) mod N$

Initialization: T = $R^2 mod N$, $a_R = a \cdot R mod N$ where $R = 2^{n+2}$

1) $s_0 = CSMMM(T, m, N)$
2) $s_1 = CSMMM(s_0, s_0, N)$
3) **For** $i = t - 2$ **to** 1
   a) **If** $e_i = 1$ **then**
      i) $s_0 = CSMMM(s_0, s_1, N)$
      ii) $s_1 = CSMMM(s_1, s_1, N)$
   b) **else**
      i) $s_1 = CSMMM(s_0, s_1, N)$
      ii) $s_0 = CSMMM(s_0, s_0, N)$
4) $s_1 = CSMMM(s_0, s_1, N)$
5) $s_0 = CSMMM(s_0, s_0, N)$
6) $s_1 = CSMMM(s_1 + a_R, 1, N)$
7) $s_0 = CSMMM(s_0 + a_R, 1, N)$
8) **If** (Loop Counter $i$ and exponent $e$ are not modified) **then** return $(s_0, s_1)$ **else** return error

The proposed algorithm has a initialization stage where the values T = $R^2 mod N$ and $a_R = a \cdot R mod N$ are computed. This stage calculations are not performed in every execution of the FSPAMME algorithm since all its values can be considered constant. Note that the all values are in Montgomery format, meaning that there is an R mod N factor in every modulus (e.x. $s_1 = s_0 \cdot s_1 \cdot R mod N$). This factor is eliminated in steps 6 and 7 where there is an MMM operation of each variable with 1.

## V. Proposed Hardware Architectures

### A. Proposed C-S Montgomery Modular Multiplier

Using the proposed CSMMM algorithm, a Montgomery modular multiplication architecture can be constructed as shown in Fig. 1. All data values are described as two $n+1$ bit values, in Carry and Save representation. The precomputed value Y+N or Y is chosen as input to a 4 to 2 Carry-Save adder (represented as two full adder structures in Fig. 1). Each full adder structure is a series of n+1 full adders arranged in parallel. There is a X reconstruction unit that is responsible for generating the control signal $X_R[k]$ of the CSMMM algorithm. This signal in the proposed architecture is generated using one 1-bit 3 to 2 full adder unit. The Carry bit generated through this full adder is stored in a D Flip Flop (DFF) and is used as input in the consecutive algorithmic round. Note, that although the utilized values in the CSMMM have n+2 bit length, the signals of the CSMMU architecture have n+1 bit length. The calculations of all the MSbit ($n$+1 bit) values can omitted since this bit is alway equal to zero. This fact can be proved when considering the MMM constrains described in section II. Each outcome of the MMM or CSMMM algorithm is smaller than 2N ($A < 2N$) therefore its bit length would be at most $n$ and before the final division by 2 it would be at most $n+1$ bits. Similarly, the utilized registers for storing each round's carry and save outputs need to have n+1 bit length. Before storing a value in each of those registers, a one bit right shifting operation is performed on the input value (the LSbit is ignored and the $n+1$ bit is cleared).

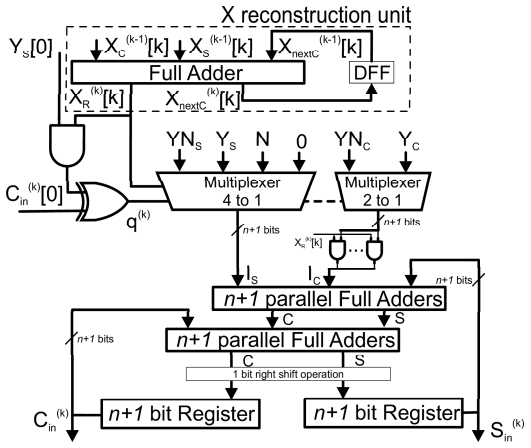


Fig. 1. The proposed C-S Montgomery Modular Multiplication Architecture (CSMMU)

### B. Proposed FA-SPA Modular Exponentiation architecture

Using the CSMMU proposed architecture of Fig 1 the FA-SPA resistant Montgomery modular exponentiation unit can be constructed. Observing FSPAMME algorithm, reveals that there are several algorithmic steps that can be performed in parallel. Parallelism can be applied on steps 3ai and 3aii, steps 3bi and 3bii, steps 4 and 5 as well as steps 6 and 7. Furthermore, one part of the step pair always performs Montgomery modular multiplication between $s_0$ and $s_1$ in parallel with the other part of the step pair performing Montgomery modular squaring either of $s_0$ or $s_1$. We assign each parallel operation in a CS Montgomery modular multiplication unit (CSMMU), like the one described in Fig 1. Therefore, only two such units are required for all necessary calculations of the FSPAMME algorithm. A problem can occur using this approach and it has to do with the correct inputs in the Montgomery squaring unit. Those inputs are taken either from the output of the same CSMMU or the output of the multiplication CSMMU. The correct input choice of the squaring CSMMU is not related only to the i-th bit of $e$ in round $i$ but it also has to do with the operations performed in the previous algorithmic round (round i-1) and therefore it is also related to the (i-1)th $e$ bit. Thus, in the proposed architecture, the data path control in squaring CSMMU is related to the value difference between the ith and (i-1)th bit of $e$ and is estimated through a XOR gate. This design approach may increase the resistance in SCA of the proposed exponentiation architecture since the connection between the key bits and the data path is not as straightforward as the original approach. The resulting FA SPA resistant modular exponentiation architecture is described in Fig. 2.

## VI. Performance

The proposed designs (CSMMU and FASPAMEA) can be effectively used for the design of an RSA unit that is resistant to FA and SPA. An RSA can be obtained by directly employing the FASPAMEA design using a small exponent (e.x. $e = 2^{16} + 1$) or utilizing FASPAMEA as part of CRT based RSA as proposed in [4], [5]. The CSMMU needs $n+2$ clock cycles to come up with a result. The FSPAMME algorithm is concluded after $t-2$ algorithmic

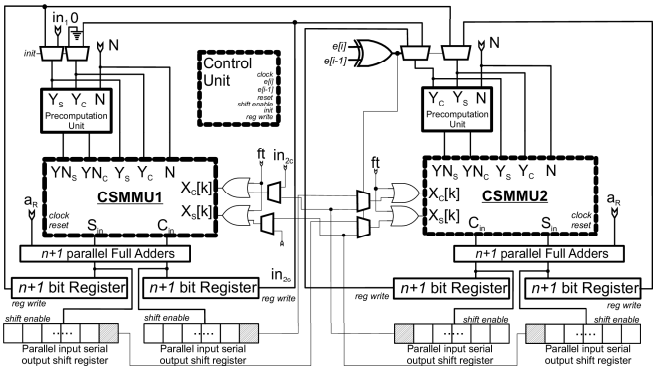


Fig. 2. The proposed FA-SPA resistant modular exponentiation architecture (FASPAMEA)

rounds where the CSMMM algorithm is executed in parallel two times in each round ($t$ is the bit length of $e$). Also, in FSPAMME, 2 execution of CSMMM are needed for initialization (FSPAMME steps 1 and 2) and 2 parallel CSMMM executions for post processing calculations (FSPAMME parallel steps 4 - 5 and 6 - 7). As a result, the total number of clock cycles for one modular exponentiation using FASPAMEA is $(t+2)\cdot(n+2)$.

Both the CSMMU and FASPAMEA proposed architectures where realized in FPGA technology (xilinx virtex 5) using VHDL language for $n = 1024$ (a typical bit length with acceptable security level for RSA) and measurements in chip covered Area (FPGA slices) and clock frequency (MHz) were taken. Those measurements are presented in Tables I and II and are compared with other similar Montgomery multiplier designs. Note that in all throughput values were calculated for small exponent RSA ($e = 2^{16}+1$) where $t = 17$.

TABLE I
MONTGOMERY MODULAR MULTIPLIER COMPARISONS ($n = 1024$)

| Arch. | Technology | Area (slices) | Freq. (MHz) |
|---|---|---|---|
| proposed | XC5VLX50T | 1793 | 180 |
| [14] | XC2V3000 | 8000 | 219 |
| [3] | XC2V3000 | 11520 | 111 |
| [13] | XC2V3000 | 3611 | 129 |

As it can be observed from Table I the proposed CSMMU architecture is very advantageous against all other compared designs. It has the smallest chip covered area of all architectures and high maximum frequency. The only design that has a higher frequency value is the design of [14]. However, this design's chip covered area is about 4.5 times bigger than the proposed CSMMU.

TABLE II
MODULAR EXPONENTIATION COMPARISONS FOR $n = 1024$

| Arch. | Technology | Area (slices) | Freq. (MHz) | Throughput | FA-SPA |
|---|---|---|---|---|---|
| prop. | XC5VLX50T | 7158 | 274 | 14.5 Mpbs | Yes |
| [14] | XC2V3000 | 12537 | 152.5 | 8.44 Mbps | No |
| [3] | XC2V6000 | 23208 | 96 | 4.79 Mbps | No |
| [13] | XC2V3000 | 7873 | 129 | 7.17 Mbps | No |
| [15] | SPARTAN-3 | 1679 | 16 | not reported | No |

Similar results can be extracted from Table II about the proposed FASPAMME architecture. The proposed design has the highest throughput, frequency values and the smallest chip covered area of all architectures with the exception of the sequential architecture in [15]. This architecture has a very constrained chip covered area but can operate on very low frequency rates (16 MHz) whereas the proposed FASPAMEA design is about 17 times faster. Note also, that FASPAMME is the only design that can be considered FA-SPA resistant. It can be concluded that the use of Montgomery multiplication with Carry Save logic in FA-SPA resistant RSA designs can result in chip covered area and speed efficient architectures.

## VII. CONCLUSION

FA and SPA resistance is an important feature in an RSA hardware module. However, this feature is considered expensive in computational cost. In this paper, we tried to change this belief by proposing an algorithm for FA and SPA resistant modular exponentiation designed for RSA that is based on an optimized version of Carry Save Montgomery modular multiplication algorithm. Using those algorithms, we proposed a Montgomery modular multiplication architecture and used it to design a FA and SPA resistant modular exponentiation architecture. Realizing the proposed architectures in FPGA technology and extracting implementation-synthesis results, revealed that FA and SPA resistant RSA design through the use of Montgomery multiplication algorithm is not only feasible but also very advantageous compared with other similar designs.

## REFERENCES


[1] A. J. Menezes, P. C. V. Oorschot, S. A. Vanstone, and R. L. Rivest, "Handbook of applied cryptography," 2001.

[2] C. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, 1999.

[3] C. McIvor, M. McLoone, and J. McCanny, "Modified montgomery modular multiplication and rsa exponentiation techniques," *IEE Proceedings - Computers and Digital Techniques*, vol. 151, no. 6, pp. 402–408, 2004.

[4] C. Giraud, "An rsa implementation resistant to fault attacks and to simple power analysis," *IEEE Transactions on Computers*, vol. 55, no. 9, pp. 1116–1120, 2006.

[5] D. Vigilant, "Rsa with crt: A new cost-effective solution to thwart fault attacks," in *CHES*, ser. Lecture Notes in Computer Science, E. Oswald and P. Rohatgi, Eds., vol. 5154. Springer, 2008, pp. 130–145.

[6] M. Joye and S.-M. Yen, "The montgomery powering ladder," in *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK: Springer-Verlag, 2003, pp. 291–302.

[7] C. H. Kim and J.-J. Quisquater, "Fault attacks for crt based rsa: New attacks, new results, and new countermeasures," in *WISTP*, ser. Lecture Notes in Computer Science, D. Sauveron, C. Markantonakis, A. Bilas, and J.-J. Quisquater, Eds., vol. 4462. Springer, 2007, pp. 215–228.

[8] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521.

[9] G. Hachez and J.-J. Quisquater, "Montgomery exponentiation with no final subtractions: Improved results," in *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK: Springer-Verlag, 2000, pp. 293–301.

[10] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology Proceedings of Crypto 99*. Springer-Verlag, 1999, pp. 388–397.

[11] K. Bhattacharya and N. Ranganathan, "A linear programming formulation for security-aware gate sizing," in *GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI*. New York, NY, USA: ACM, 2008, pp. 273–278.

[12] K. Tiri and I. Verbauwhede, "A digital design flow for secure integrated circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1197–1208, 2006.

[13] A. P. Fournaris and O. G. Koufopavlou, "A new rsa encryption architecture and hardware implementation based on optimized montgomery multiplication," in *ISCAS (5)*. IEEE, 2005, pp. 4645–4648.

[14] M.-D. Shieh, J.-H. Chen, H.-H. Wu, and W.-C. Lin, "A new modular exponentiation architecture for efficient design of rsa cryptosystem," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 9, pp. 1151–1161, 2008.

[15] N. Nedjah and L. de Macedo Mourelle, "Three hardware architectures for the binary modular exponentiation: Sequential, parallel and systolic," *IEEE Trans. Circ. and Syst. I: Regular Papers*, vol. 53, no. 3, pp. 627–633, 2006.