MODULAR MULTIPLICATION ALGORITHM FOR PKC

Diego Viot, Rodolfo Aurélio, Helano Castro and Jardel Silveira
Universidade Federal do Ceará, LESC
Campus do PICI S/N, Bloco 723
Fortaleza, CE – Brasil
E-mail: {diego, rodolfo, helano, jardel}@lesc.ufc.br

ABSTRACT

Side-channel attack is any attack based on information gained from the physical implementation of a cryptosystem. Private-key operations need a due amount of time to be performed. Attackers are able to break these systems by unmistakably measuring such time, what characterizes a timing attack, a kind of side-channel attack. Some algorithms for public-key encryption, like RSA, exhaustively use the mathematical operation of modular exponentiation, which has the modular multiplication as basis. Thus, in this paper we compare two different methods of modular multiplication by their applicabilities in timing attack resistant integrated circuits and efficiencies. Furthermore, we implemented three algorithms, analyze its faculty for preventing timing attacks and provide adequate hardware implementations.

1. INTRODUCTION

High level security and data integrity are indispensable in order to manage confidential information in network applications, such as e-mail and e-banking, and in mobile applications, such as embedded security and credit card operations. Public Key Cryptography (PKC) bases the security on actual digital information systems. PKC uses two keys, a "public key" and a "private key", to implement an encryption algorithm that doesn't require two parties to first start an exchange of a secret key in order to conduct secure communications.

RSA, which was invented by Rivest, Shamir and Adleman in 1978 [1], is the most used pattern in PKC. The algorithm's principle based on the construction of both public and private keys using big prime numbers randomly generated. Any ciphered message that uses public key needs its respective private key to be deciphered. To achieve this, the following mathematical formulas are used:

$$C = M^{E} \bmod N , \qquad (1)$$

$$T = C^{D} \bmod N , \qquad (2)$$

where M is the original message, E is the private key, C is the ciphered message and D is the public key.

Because of the magnitude of intermediate results of exponentiation using large keys, modular exponentiation is not done using the most straightforward method; modular multiplication is applied repeatedly instead. A modular multiplication, such as, $A \cdot B \bmod M$, where A and B are the operands and M is the modulus, can be performed in two different ways: multiplying then reducing or interleaving the multiplication and the reduction steps.

This paper's purpose is to analyze Montgomery's modular multiplication algorithm [2] as a base for modular exponentiation and to compare it to the method of multiplying then reducing. Throughout the subsequent sections, we will revise some important topics, describe the implemented algorithms and analyze its behavior under timing attacks as well as its hardware implementation feasibility.

2. SIDE-CHANNEL ANALYSIS (TIMING ATTACK)

Mathematical models are used to describe conventional cryptographic systems. However, some events occurring during the cryptography's implementation cannot be exactly described by mathematical tools. It permits that attacks may infer the secret key by using techniques like reverse engineering and micro probing [3].

Side-channel attacks are attacks that are based on side-channel information. Side-channel information is information that can be retrieved from the encryption device that is neither the plaintext to be encrypted nor the ciphertext resulting from the encryption process.

Modern encryption devices have plaintext, ciphertext and additional inputs and outputs. These devices produce information about the length of time that operations take, radiation of various sorts and power consumption statistics [4]. Often the voltage inputs can be modified to cause predictable outcomes. By making use of this information and other known cryptanalytic techniques, attackers may recover the key the device is using.

Reasons like bypassing unnecessary operations, branching and conditional statements, RAM cache hits and processor instructions that do not run in fixed time are responsible for different amount of time to process different inputs. Timing attacks are based on measuring and computing variances of these amounts of time. The attack can be tailored to work with virtually any implementation that does not run in fixed time and assumes that the attacker knows the design of the target system (although in practice this could probably be inferred from timing information). Details about how general attacks happen, including the statistical models used, are shown in [4].

3. MODULAR MULTIPLICATION

The modular multiplication is used to perform modular exponentiations, which, in their turn, are used by several public-key cryptosystems. The performance of public-key cryptosystems is primarily determined by the implementation's efficiency of the modular exponentiation. So, consequently, modular multiplication is an important factor in these systems.

There are two basic methods of performing a modular multiplication: multiply-then-reduce and interleaved multiplication and reduction.

3.1. Multiply-then-Reduce Method

The multiply-then-reduce method consists of first computing the product then reducing it with respect to the given modulus. Insofar, as it is an elementary multiplying, many algorithms, like add-and-shift, for example, may be used. The most popular multiplication methods are Karatsuba-Ofman's method and Booth's method [5].

A modular reduction is simply the computation of the remainder of an integer division. The naive sequential algorithms successively shifts and subtracts the modulus until a non-negative and smaller value than the modulus remainder is found. More complex algorithms, which minimize the number of compare-and-subtract steps and, consequently, maximize the complete operation's speed, may also be implemented. As an example, we can quote the multiply-then-reduce Barrett's algorithm. Modular reduction is always dependent of inputs and intermediate results, which is the main cause of timing variations in a modular operation.

Modular multiplications consume most of the total modular exponentiation time. As a result of timing variations of reduction steps, multiply-then-reduce method is more vulnerable to timing attacks. Three multiplying algorithms were used for implementing the modular multiplication, as described in subsequent sections.

3.1.1 The Add-and-Shift Algorithm

The add-and-shift algorithm of multiplication is performed using only add and shift operations. The partial product starts in zero and then each multiplier's bit is processed at a time. The multiplicand is added to partial product if that bit is set and, at the ending of this process, the partial product is right-shifted.

Here, the reduction step is implemented with successive subtractions until the result is less than the modulus.

```
algorithm ModularMult(A, B, M, n)
   const C := 2^n mod M;
   int R := 0;
   R := Montgomery(A, B, M);
   return Montgomery(R, C, M);
end ModularMult.
```

```
algorithm Montgomery(A, B, M)
      int R = 0;
      1: for i= 0 to n-1
      2:    R = R + a_i×B;
      3:    if r_0 = 0 then
      4:       R = R div 2
      5:    else
      6:       R = (R + M) div 2;
      return R;
end Montgomery.
```

Figure 1: Montgomery's modular multiplication algorithm

3.1.2 The Booth's Algorithm

The Booth's algorithm is also performed with partial products, but it uses several partial product generators together with several adders that operate in parallel. Each partial product obtained is shifted left or right depending on whether the starting bit was the less or the most significant and added up. The number of partial products generated is bound above by the size of the multiplier operand. So, once the sum of the partial products is obtained, the rest of this sum is finally the result of the multiplication.

Likewise the previous algorithm, the reduction step will be implemented with successive subtractions.

3.2. Interleaved Multiplication and Reduction Method

The idea of interleaving multiplication and reduction steps is very simple: the first operand is multiplied with the second operand bitwise and added to the intermediate result. Then the intermediate result is reduced with respect to the modulus.

The naive interleaved multiplication and reduction method is implemented with subtractors, just like is done with multiply-then-reduce method.

However, this is not the best way to yield results with no timing variations; insofar it has the same above method's problem of depending on the number of compare-and-subtract steps. More sophisticated methods are also widely implemented. The most popular are Montgomery's [2] and Brickell's [6] methods. Montgomery's algorithm will be described in the section bellow.

3.2.1 The Montgomery's Algorithm

The Montgomery's algorithm of multiplication of is a very ingenious method of calculating the modular multiplication. It replaces the division by adding a shift and modulate, if necessary, which are much faster to your computer. Furthermore, the algorithm is best suited for implementation in hardware (FPGA / ASIC).

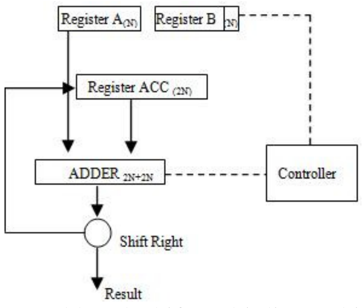

Figure 2: Add-and-Shift multiplier architecture

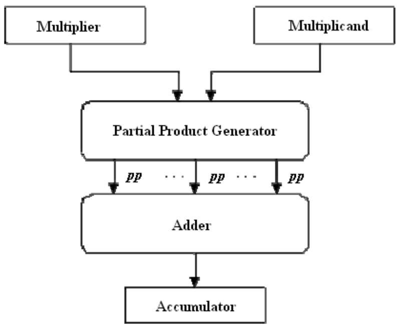

Figure 3: Booth's multiplier architecture

The multiplication of modular Montgomery calculates MM(X,Y) = XYR-1 mod m, where m is an integer in the interval $2^{n-1} \leq m < 2^n$ such that MDC(m,R) = 1 (this requirements is satisfied if m odd).

Following [5], we implemented a multiplication based on an iterative adder-accumulator for the generated partial products, as shown in the algorithm of Figure 1.

Nowadays, many other implementations of Montgomery's modular multiplication are known. Some of them may be found in [7].

4. HARDWARE IMPLEMENTATION

This section describes essentially the synthesis and simulation of the targeted architecture coded with hardware description languages. The design capture of the modular multiplication's architectures using VHDL and Verilog was done for 32, 128 and 1024 bits.

Consisting of an adder, an accumulator, a shifter and the controller, as shown in Figure 2, a naive add-and-shift architecture was implemented.

Following a similar flow, Booth's architecture was implemented using partial product generators, adders and an accumulator, as shown in Figure 3.

According to [5], we used a given architecture (shown in Figure 4) to build our own VHDL source code. Shift registers, adders, multiplexers, registers and a controller were used. Except for the controller, which does the management work, all other blocks are responsible for every mathematical computation.

This algorithm computes the product of two integers modulus a third one without performing division by M. It yields the reduced product using a series of addition.

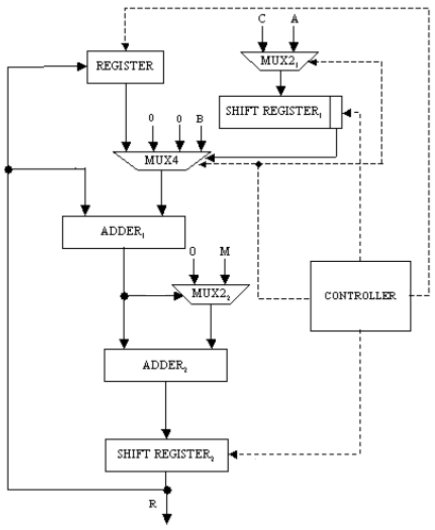

Figure 4: Montgomery's modular multiplier architecture

5. IMPLEMENTATION RESULTS AND COMPARISON

Implementation results of our three architectures, which were implemented on a Xilinx FPGA XC4VFX12, are shown in the following tables. Table 1 compares the timing operation, showing the maximum workable frequencies and minimum workable periods; Table 2 compares the device utilization by the number of used FPGA configurable logic block slices and its representative percentage. For each architecture three bit lengths (32, 128 and 1024) were tested. Add-and-shift method, Booth's and Montgomery's methods are represented in the tables by A&S, Booth and MG, respectively. In order to synthesize and simulate we used Xilinx ISE 9.2 and Cadence NC-Sim software.

As major part of this work, timing analysis was done comparing results of two basic modular multiplications. Utilizing the implemented 32 bits architectures, the response times will be measured to present its variation (or its lack of variation). However, in cryptosystems, operands are quite large (as 1024 bits) as they represent blocks of text. So this comparison will be barely useful to illustrate the response time variation.

First, we applied operands A = 13 and B = 25 and modulus M = 97. Then, we applied A = 111, B = 86 and M = 113. The results are given in Table 3. Note that in the first modular multiplication the reduction step will subtract less times than in the second modular multiplication.

Considering the results above, we perceived that the Booth's method, in spite of slower response, is better than add-and-shift method, because of its less FPGA usage. However, due to the variations of response times, Montgomery's method (and so the interleaved multiplication and reduction method) is better than the others for public-key encryption applications.

Table 1: Maximum frequency and minimum period depending on the bit length.

Method	Operand size	Max. Freq.	Min. Period
A&S	32	140MHz	7.135ns
	128	62MHz	16.080ns
	1024	10MHz	98.811ns
Booth	32	221MHz	4.511ns
	128	113MHz	8.783ns
	1024	20MHz	48.656ns
MG	32	532MHz	1.878ns
	128	465MHz	2.150ns
	1024	303MHz	3.298ns

Table 2: FPGA configurable logic block slices' usage (out of 5472) depending on the bit length.

Method	Operand size	# Slices	Percentage
A&S	32	260	4%
	128	1068	19%
	1024	8190	149%
Booth	32	106	1%
	128	433	7%
	1024	3496	63%
MG	32	185	3%
	128	654	11%
	1024	5038	92%

Table 3: Response times of modular multiplications on our three architectures with common clock cycles.

Method	Mod. Mult.	Time
A&S	$13 \cdot 25 \bmod 97$	65ns
	$111 \cdot 86 \bmod 113$	450ns
Booth	$13 \cdot 25 \bmod 97$	216ns
	$111 \cdot 86 \bmod 113$	702ns
MG	$13 \cdot 25 \bmod 97$	1174ns
	$111 \cdot 86 \bmod 113$	1174ns

Is important to realize that the Montgomery's algorithm is relatively not good for small bit lengths, but behaves stably for large bit lengths. In contrast, multiply-then-reduce method behaves extremely badly for large bit lengths: sometimes is quick and sometimes bring results exceptionally slowly. Besides the problem of becoming a slow calculation, this inconstancy causes the variation in response time, that is the information used by attackers for a timing attacks.

6. CONCLUSION AND FURTHER WORK

We presented three modular multiplication methods that bring different implementation results. After analyzing the results presented in the last section we conclude that, under the side-channel attacks perspective (timing attacks in particular), the Montgomery's method came across being very resistant.

As a future work we will implement a complete circuit of RSA encryption algorithm using the Montgomery's algorithm described above. We are also inclined to make improvements in our design for performance and area optimization.

7. ACKNOWLEDGEMENTS

The authors are grateful to the Funcap's founding support through its post-graduating scholarship program and also to Xilinx and Cadence for EDA tools support, provided by their University programs.

8. REFERENCES

[1] R. L. Riviest, A. Shamir, ans L. M. Adleman. A method for obtaining digital signatures and public-keys cryptosystems. *Communication of the ACM,* 21(2):120-126, 1978.

[2] P. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[3] C. Lu, S. Tseng, and S. Huang, "A Secure Modular Exponential Algorithm Resists to Power, Timing, C Safe Error and M Safe Error Attacks", in Proc. AINA, 2005, pp.151-154.

[4] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems"', *Proc. of CYRPTO '97*, Springer LNCS vol. 1109, pp. 104-113, 1997.

[5] N. Nedjah and L. Mourelle. A review of modular multiplication methods and respective hardware implementations. Informatica, 30 :111–130, 2006.

[6] Brickell, E. F., *A survey of hardware implementation of RSA*, In G. Brassard, ed., Advances in Cryptology, Proceedings of CRYPTO'98, Lecture Notes in Computer Science 435:368-370, Springer-Verlag, 1989.

[7] Koç Ç. K.; Acar T; Kalisky Jr, B.S.. – Analyzing and Comparing Montgomery Multiplication Algorithms; IEEE Micro, v.16 n.3, p.26-33, June 1996.