Redundant Modular Reduction Algorithms

Vincent Dupaquis, Alexandre Venelli

Redundant Modular Reduction Algorithms

Vincent Dupaquis and Alexandre Venelli

Inside Secure
Avenue Victoire, 13790 Rousset, France
{vdupaquis,avenelli}@insidefr.com

Abstract. We present modular reduction algorithms over finite fields of large characteristic that allow the use of redundant modular arithmetic. This technique provides constant time reduction algorithms. Moreover, it can also be used to strengthen the differential side-channel resistance of asymmetric cryptosystems. We propose modifications to the classic Montgomery and Barrett reduction algorithms in order to have efficient and resistant modular reduction methods. Our algorithms are called dynamic redundant reductions as random masks are intrinsically added within each reduction for a small overhead. This property is useful in order to thwart recent refined attacks on public key algorithms.

1 Introduction

Modular reduction is at the heart of many asymmetric cryptosystems. Its principle is to evaluate the remainder of the integer division x/m. However, the division of two large multi-precision integers is very costly. Modular reduction algorithms were proposed in order to compute efficiently a remainder. Barrett reduction [5] and Montgomery reduction [15] are the two main methods. Both algorithms have a pre-computational step where either the inverse of the modulus or its reciprocal is computed. If the modulus is fixed amongst many operations, this pre-computed value is used in order to efficiently obtain a modular reduction. In asymmetric cryptosystems, the modulus is generally fixed at the very beginning. These techniques are then extremely efficient. Note that we leave out of our study the use of interleaved reduction and multiplication.

The implementation of cryptography algorithms on embedded devices is particularly sensitive to side-channel attacks. An attacker is often able to recover secret information from the device simply by monitoring the timing variations [11] or the power consumption [12]. Side-channel analysis include two main families of attacks: simple side-channel analysis (SSCA) and differential side-channel analysis (DSCA).

In this paper, we aim at protecting asymmetric cryptosystems against DSCA. Particularly, we focus on securing the exponentiation algorithm when the exponent is secret. Messerges *et al.* in [14] first detailed DSCA attacks applied to a modular exponentiation. Powerful attacks on public key algorithms were also proposed recently by Amiel *et al.* in [2, 1]. These attacks can combine information obtained from SSCA, DSCA or fault analysis. There are typically two

families of countermeasures against DSCA: randomized addition chains type or data randomization type. This second family is very interesting as it provides a countermeasure independent of the choice of the algorithm. Well known protections consist in message blinding [14], exponent blinding [11] and exponent splitting methods [7]. We propose here an alternative DSCA countermeasure that thwarts most of the attacks published in the literature.

We study in detail a technique called redundant modular arithmetic that allows integers modulo m to be kept modulo m plus some random multiples of m such that their representation is not unique. This idea has been introduced before in order to avoid timing attacks [20]. However, it can be improved to thwart also differential side-channel analysis. Golić and Tymen [10] proposed a differential side-channel countermeasure, based on the idea of redundant arithmetic, in the context of a masked Advanced Encryption Standard (AES) implementation. A brief study of Montgomery redundant arithmetic as a differential side-channel countermeasure is presented by Smart *et al.* in [18].

We extend this work to a so-called dynamic redundant modular arithmetic applied to Barrett's and Montgomery's reduction. Compared to [18], our algorithms offer flexibility in their usage as a randomization parameter can be adjusted. Our solution also allows dynamic randomization inside the exponentiation algorithm whereas other classical countermeasures often use only one randomization at the beginning of the exponentiation algorithm. This property can be useful in order to have a protection against the attack of Amiel *et al.* in [1] or even combined attacks as presented in [3].

The remainder of this paper is structured as follows. Section 2 describes the Montgomery and Barrett modular reduction algorithms. In Section 3, we detail the static redundant arithmetic method previously introduced in [18]. Then, we propose dynamic redundant reduction algorithms based on Montgomery's and Barrett's reductions in Section 4. We evaluate the performance and the security of our propositions in Section 5. Finally, Section 6 concludes the paper.

2 Modular Reduction Algorithms

We first introduce some notations. Long integers are represented as arrays of w-bit digits. Generally, one choose w as the word-size of the processor. The bit length of the integers is noted l and n is the number of digits necessary to store them, hence $n = \lceil l/w \rceil$. A long integer is then noted as $u = (u_{n-1}, \ldots, u_0)_b$ with $0 \leq u_i < b$ and $b = 2^w$. We note that, on a processor which word-size is w, the division by b or a power of b is simply a right shift, hence virtually free. Let $m = (m_{n-1}, \ldots, m_0)_b$ be a prime modulus of size n, *i.e.* n digits in the b-basis representation. Let u and v be two integers strictly lower than m, hence of size at most n. Let $x = (x_{2n-1}, \ldots, x_0)_b = uv < m^2$ be the integer to reduce modulo m.

2.1 Montgomery Reduction

The Montgomery reduction method [15] consists in using divisions by a power of b instead of multi-precision divisions. Let $R > m$ an integer coprime to m such that operations modulo R are easy to compute. A classical choice is $R = b^n$. In this method, integers $u < m$ are represented as a m-residues with respect to R, *i.e.* $uR \bmod m$. This representation of an integer is often called a Montgomery form. The Montgomery reduction of u is then defined as $uR^{-1} \bmod m$ where R^{-1} is the inverse of R modulo m. This reduction supposes that integers are in the Montgomery form. Take two integers $u < m$ and $v < m$. Consider their transformation in Montgomery form $uR \bmod m$ and $vR \bmod m$. Their multiplication gives $x = uvR^2$. Then, using Algorithm 1, one obtains $x = uvR \bmod m$ which is still in Montgomery form. This method uses a pre-computed value $\beta = -m^{-1} \bmod R$. Note also that the reduction requires, at most, only one final subtraction by m. Let $\mathsf{MontRed}(x, m, R, \beta)$ be the Montgomery reduction algorithm presented in Algorithm 1.

Algorithm 1 Montgomery reduction algorithm

Input: positive integers $x = (x_{2n-1}, \ldots, x_0)_b, m = (m_{n-1}, \ldots, m_0)_b$ and $\beta = -m^{-1} \bmod R$ where $R = b^n$, $\gcd(b, m) = 1$ and $x < mR$
Output: $xR^{-1} \bmod m$
1: $s_1 \leftarrow x \bmod R,\ s_2 \leftarrow \beta s_1 \bmod R,\ s_3 \leftarrow m s_2$
2: $t \leftarrow (x + s_3)/R$
3: **if** $(t \geq m)$ **then**
4: $\quad t \leftarrow t - m$
5: **end if**
6: **return** t

2.2 Barrett Reduction

Introduced by Barrett [5], this method is based on the idea of fixed point arithmetic. The principle is to estimate the quotient x/m with operations that can either be pre-computed or are less expensive than a multi-precision division. The remainder r of x modulo m is equal to $r = x - m \lfloor x/m \rfloor$. Using the fact that divisions by a power of b are virtually free, we have:

$$r = x - m \left\lfloor \frac{\frac{x}{b^{n-1}} \frac{b^{2n}}{m}}{b^{n+1}} \right\rfloor = x - m \left\lfloor \frac{\frac{x}{b^{n-1}} \mu}{b^{n+1}} \right\rfloor$$

with $\mu = \left\lfloor \frac{b^{2n}}{m} \right\rfloor$ a pre-calculated value that depends on the modulus. Let $\hat{q}$ be the estimation of the quotient of x/m. Barrett improves further the reduction using only partial multi-precision multiplication when needed. The estimate $\hat{r}$ of the remainder of x modulo m is:

$$\hat{r} = (x \bmod b^{n+1} - (m\hat{q} \bmod b^{n+1})) \bmod b^{n+1}.$$

This estimation implies that at most two subtractions of m are required to obtain the correct remainder r. Barrett's algorithm is described in Algorithm 2. Note that in this algorithm, the estimated quotient $\hat{q}$ corresponds to the variable q_3 and the estimated remainder $\hat{r}$ is computed with the operation $r_1 - r_2$ in Line 2.

Algorithm 2 Barrett reduction algorithm

Input: positive integers $x = (x_{2n-1}, \ldots, x_0)_b, m = (m_{n-1}, \ldots, m_0)_b$ and $\mu = \lfloor b^{2n}/m \rfloor$
Output: $x \bmod m$
1: $q_1 \leftarrow \lfloor x/b^{n-1} \rfloor,\ q_2 \leftarrow \mu q_1,\ q_3 \leftarrow \lfloor q_2/b^{n+1} \rfloor$
2: $r_1 \leftarrow x \bmod b^{n+1},\ r_2 \leftarrow m q_3 \bmod b^{n+1},\ r \leftarrow r_1 - r_2$
3: **if** $(r \leq 0)$ **then**
4: $\quad r \leftarrow r + b^{n+1}$
5: **end if**
6: **while** $(r \geq m)$ **do**
7: $\quad r \leftarrow r - m$
8: **end while**
9: **return** r

3 Static Redundant Modular Arithmetic

Redundant field representation can form an interesting defense against differential side-channel attacks as a given element can have different representations. We can cite the recent work by Smart *et al.* [18] on this topic. The authors briefly present a redundant Montgomery arithmetic. If one wants to work with integers modulo m, a standard representation is to take elements in $\mathbb{Z}/m\mathbb{Z} = \{0, \ldots, m-1\}$. The principle of redundant arithmetic is to consider elements in the range $\{0, \ldots, C-1\}$ where $C > m$ and keep integers modulo m within this range.

We recall this method that can be denoted as static redundant modular arithmetic.

Let $C = cm$ where $c > 1$ is an integer coprime to m. Instead of working in $\mathbb{Z}/m\mathbb{Z}$, we work in $\mathbb{Z}/(cm)\mathbb{Z}$, *i.e.* modulo C. Let $R_C = b^j$ with $j > n$ a certain integer such that $R_C > C$. Let $\beta_C = -C^{-1} \bmod R_C$ a pre-computed value similar to β defined in § 2.1. The Montgomery form of an element u modulo C becomes $uR_C \bmod C$. We apply this method in a classical modular exponentiation algorithm (see Algorithm 3).

We call this method static redundant modular arithmetic as only one random mask k is applied at the beginning of the algorithm. In the next section, we present two propositions of dynamic redundant modular reduction algorithms for the methods of Montgomery and Barrett.

Algorithm 3 Square-and-multiply exponentiation using static redundant Montgomery arithmetic

Input: positive integers $e = (e_{l-1}, \ldots, e_0)_2, x, m, C, \beta_C$ and R_C
Output: $x^e \bmod m$
1: $X \leftarrow x + km$, where k is a random integer
2: $R_0 \leftarrow R_C$
3: $R_1 \leftarrow XR_C \bmod C$
4: **for** $i = l - 1$ down to 0 **do**
5: $\quad R_0 \leftarrow \mathsf{MontRed}(R_0^2, C, R_C, \beta_C)$
6: $\quad$ **if** $(e_i = 1)$ **then**
7: $\quad\quad R_0 \leftarrow \mathsf{MontRed}(R_0 R_1, C, R_C, \beta_C)$
8: $\quad$ **end if**
9: **end for**
10: $R_0 \leftarrow R_0 R_C^{-1} \bmod m$
11: **return** R_0

4 Dynamic Redundant Modular Arithmetic Propositions

We propose modular reduction algorithms that are slight modifications of Montgomery's and Barrett's techniques. Our methods offer a so-called dynamic redundant modular arithmetic in which random masks are refreshed intrinsically within the reduction method.

4.1 Dynamic Redundant Montgomery Reduction

Consider we want to reduce an integer $x = uv$ modulo m with $u < m$ and $v < m$ two integers modulo m. Hence, if m is a n-word integer, we want to reduce a $2n$-word integer x. Working with redundant modular arithmetic, several multiples of the modulo m can be added to an integer. Thus, its size will grow depending on the number of multiples considered. If we have a $(2n+2i)$-word integer at the input of the reduction algorithm, we need a $(n+i)$-word output so that further multiplications between reduced elements can be computed.

We first look in details at the operations of the Montgomery reduction (Algorithm 1). Recall that $\beta = -m^{-1} \bmod R$ is a pre-computed value and R is a power of b, generally $R = b^n$. We know from [13, Fact 14.29 and Note 14.30] that $(x + m(x\beta \bmod R))/R = (xR^{-1} \bmod m) + \epsilon m$ with $\epsilon \in \{0, 1\}$ at the end of the reduction. If we develop the formula, we have:

$$\begin{aligned}(x + m(x\beta \bmod R))/R &= (x + m(x\beta - k_1 R))/R &(1)\\ &= (x + x(-1 + k_2 R) - k_1 Rm)/R &(2)\\ &= k_2 x - k_1 m, &(3)\end{aligned}$$

for k_1, k_2 some integers. In our context, we want to have ϵ greater than 1 in order to have several multiples of the modulus at the end of the reduction. By slightly modifying the Montgomery reduction algorithm, we can achieve this

property. Consider now the following steps for a modified Montgomery reduction algorithm:

1: $s_1 \leftarrow x \bmod R$
2: $s_2 \leftarrow \beta s_1 \bmod R$
3: $s_2 \leftarrow s_2 + kR$, with k some random positive integer
4: $s_3 \leftarrow m s_2$
5: $t \leftarrow (x + s_3)/R$

We now have from Eq. (3) and Step 3 that

$$(x + m(x\beta \bmod R))/R = k_2 x - (k_1 - k)m.$$

Hence, the error at the end of the reduction becomes $k + \epsilon$. We also note that the final subtraction step of Algorithm 1 is removed so that the implementation is resistant to timing attacks. If t is the output of this modified Montgomery reduction, we have:

$$(xR^{-1} \bmod m) + km \leq t \leq (xR^{-1} \bmod m) + (k+1)m. \tag{4}$$

For a practical implementation, we need to consider the size of the operands at the input and output of the reduction algorithm. Instead of fixing the constant $R = b^n$ for a modulus of size n, we consider a larger constant $R' = b^{n+2i}$ for some integer i. We recall that the Montgomery reduction requires that the input x is such that $x < mR$. Hence, with the constant R' we can process larger integers $x < mR' < b^{2n+2i}$ which means that the output of the reduction can be integers $t < b^{n+i}$. From Eq. (4), we deduce that the random integer k needs to be chosen such that $k < b^i - 1$. We define a dynamic redundant Montgomery reduction algorithm (see Algorithm 4).

Algorithm 4 Dynamic redundant Montgomery reduction algorithm

Input: positive integers $x = (x_{2n+2i-1}, \ldots, x_0)_b, m = (m_{n-1}, \ldots, m_0)_b$ and $\beta' = -m^{-1} \bmod R'$ where $R' = b^{n+2i}$, $\gcd(b, m) = 1$, $x < mR'$ and for some integer i
Output: $t \leq (xR'^{-1} \bmod m) + (k+1)m$
1: Let k be a random integer such that $0 \leq k < b^i - 1$
2: $s_1 \leftarrow x \bmod R'$, $s_2 \leftarrow \beta' s_1 \bmod R'$
3: $s_2 \leftarrow s_2 + kR'$, $s_3 \leftarrow m s_2$
4: $t \leftarrow (x + s_3)/R'$
5: **return** t

Each reduction adds a random number of multiples of the modulus to the remainder. If a dynamic redundant reduction technique is used in an asymmetric cryptosystem such as RSA, a final normalization step is needed at the very end of the exponentiation algorithm for example in order to output a result strictly inferior to m. We note that the boundary $b^i - 1$ on the random integer k can be parametrized in the reduction algorithm. Hence, we could fix $k = 0$ and obtain a

time constant Montgomery reduction without final reduction if we want effectiveness instead of differential side-channel resistance. Let $\mathsf{DRMontRed}(x, m, R', \beta')$ be the dynamic redundant Montgomery reduction algorithm presented in Algorithm 4.

4.2 Dynamic Redundant Barrett Reduction

We know from [13, Note 14.44] that, with the original parameters of Barrett used in § 2.2, the estimated quotient $\hat{q}$ satisfies $q - 2 \leq \hat{q} \leq q$ where q is the correct quotient. It can be shown that for about 90% of the values of $x < m^2$ and m, the value of $\hat{q}$ will be equal to q and in only 1% of cases $\hat{q}$ will be 2 in error [6]. Dhem's work [9, Section 2.2.4] provides a general parametrization of the Barrett reduction. In particular, with appropriate choices, the error on the quotient can be reduced. In our context, we are interested in adding more errors to the estimated quotient so that multiples of the modulus are left in the reduced integer. However, using Dhem's parametrization to bound the errors is not interesting as, in practice, only a very small number of integers will reach the upper bound.

We first recall some notations of Dhem's work. The estimated quotient $\hat{q}$ in the reduction of x modulo m can be evaluated as:

$$\hat{q} = \left\lfloor \frac{\frac{x}{b^{n+\beta}} \frac{b^{n+\alpha}}{m}}{b^{\alpha-\beta}} \right\rfloor = \left\lfloor \frac{\frac{x}{b^{n+\beta}} \mu_\alpha}{b^{\alpha-\beta}} \right\rfloor,$$

with $\mu_\alpha = \lfloor b^{n+\alpha}/m \rfloor$ the parametrized constant used in the Barrett reduction and α, β two integers.

In Barrett's algorithm, the estimated quotient is undervalued. Hence, we need to further undervalue it in order to have a reduced integer with some multiples of the modulus left. If we have $\hat{q} = (q - k)$ the undervalued quotient for some positive integer k, then the estimated remainder will be $\hat{r} = x - m\hat{q} = x - m(q-k) = x - mq + km$. Consider the following steps for a modified Barrett reduction algorithm :

1: $q_1 \leftarrow \lfloor x/b^{n+\beta} \rfloor$
2: $q_2 \leftarrow \mu_\alpha q_1$
3: $q_3 \leftarrow \lfloor q_2/b^{\alpha-\beta} \rfloor$
4: $q_3 \leftarrow q_3 - k$, with k some random positive integer
5: $r_1 \leftarrow x \bmod b^\alpha$
6: $r_2 \leftarrow mq_3 \bmod b^\alpha$
7: $r \leftarrow r_1 - r_2$
8: **if** $(r \leq 0)$ **then**
9: $\quad r = r + b^\alpha$
10: **end if**

In practice, we note that the last conditional addition is not required on most processors that use the two's-complement system. We also remove the

final subtraction loop in Algorithm 2 as we want multiples of the modulus left in the remainder. As in the Montgomery reduction case, we consider integers x of size $2n + 2i$ to be reduced modulo m of size n. Hence, the output of the reduction algorithm needs to be at most of size $n + i$. We know from Dhem's work that the error ϵ in the estimation of the quotient is such that :

$$\epsilon \leq \lfloor 2^{n+2i-\alpha} + 2^{\beta+1} + 1 - 2^{\beta-\alpha} \rfloor.$$

We choose $\alpha = n + 2i$ and $\beta = -1$, hence the estimated quotient is, at most, undervalued by 2, *i.e.* $\epsilon \leq 2$. We do not need to minimize the error for our proposition as it would add complexity to the reduction algorithm. However the error can not be too large as for the final normalization step we need to subtract the remaining multiples of the modulus relatively quickly. If r is the output of this modified Barrett reduction, we have:

$$(x \bmod m) + km \leq r \leq (x \bmod m) + (k + 2)m. \tag{5}$$

From Eq. (5), we deduce that the random integer k needs to be defined such that $k < b^i - 2$. We simply note $\mu' = \mu_{n+2i} = \lfloor b^{2n+2i}/m \rfloor$ in the following. We define a dynamic redundant Barrett reduction algorithm (Algorithm 5). As previously, we note that the randomization can be parametrized such that if we fix $k = 0$, we obtain an efficient time constant Barrett reduction.

Algorithm 5 Dynamic redundant Barrett reduction algorithm

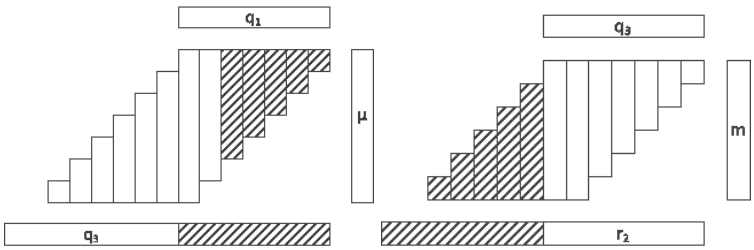
Input: positive integers $x = (x_{2n+2i-1}, \ldots, x_0)_b, m = (m_{n-1}, \ldots, m_0)_b$ and $\mu' = \lfloor b^{2n+2i}/m \rfloor$
Output: $r \leq (x \bmod m) + (k + 2)m$
1: Let k be a random integer such that $0 \leq k < b^i - 2$
2: $q_1 \leftarrow \lfloor x/b^{n-1} \rfloor$, $q_2 \leftarrow \mu' q_1$
3: $q_3 \leftarrow \lfloor q_2/b^{n+2i+1} \rfloor$, $q_3 \leftarrow q_3 - k$
4: $r_1 \leftarrow x \bmod b^{n+2i}$, $r_2 \leftarrow mq_3 \bmod b^{n+2i}$, $r \leftarrow r_1 - r_2$
5: **return** r

5 Efficiency and Security Evaluation

As previously stated, we only study reduction algorithms and leave out of our study the use of interleaved multiplication and reduction. Hence, we compare the complexity of reduction algorithms alone. Each algorithm needs a multi-precision multiplication for its intermediate results. We consider Comba's multiplication [8] as it is very interesting in embedded devices. This method is often called column-wise multiplication or product scanning method. Note that an evolution of Comba's multiplication, called hybrid multiplication, is presented in [17]. The authors combine column-wise and line-wise multiplication techniques

for better efficiency. In order to compare the performance of the algorithms, we consider the number of base multiplications, *i.e.* multiplication of two b-bit operands.

We first analyze the standard Montgomery reduction (Algorithm 1). The operation $s_2 \leftarrow \beta s_1 \bmod R$ can be computed as a partial multiplication. In fact, only the lower words of βs_1 are needed for the result s_2. We recall that $R = b^n$ with n the size of the modulus in words. The multiplicands β and s_1 are two n-words values. Instead of n^2 base multiplications, this operation can be computed with $\binom{n+1}{2} = (n^2+n)/2$ base multiplications. The next multiplication, $s_3 \leftarrow m s_2$ has to be fully computed using n^2 base multiplications. Hence the standard Montgomery reduction requires $(3n^2 + n)/2$ base multiplications.



(a) (b)

Fig. 1: Partial multiplications used in a standard Barrett reduction. Figure 1a represents the computation of q_3. Figure 1b represents the computation of r_2. The striped lines correspond to operations that are not needed for these partial multiplications.

The standard Barrett reduction (Algorithm 2) benefits also from partial multiplications. Only the highest words of the multiplication $q_2 \leftarrow \mu q_1$ are needed to compute $q_3 \leftarrow \lfloor q_2/b^{n+1} \rfloor$. As noted in [13, Note 14.45], the $n+1$ least significant words of q_2 are not needed. If $b > n$, we can compute the multiplication starting at the $n-1$ least significant word and we will have an error at most 1. The complexity is $(n+1)^2 - \binom{n}{2} = (n^2 + 5n + 2)/2$ base multiplications. The operation $r_2 \leftarrow m q_3 \bmod b^{n+1}$ can also be computed using a partial multiplication as only the $n+1$ least significant words are required. Its complexity is $\binom{n+1}{2} + n = (n^2 + 3n)/2$. Finally, the complexity of the standard Barrett reduction is $n^2 + 4n + 1$ base multiplications. The two partial multiplications used in the Barrett reduction are represented in Figure 1.

Using similar considerations as for the standard reduction algorithms, we analyze the complexity of our dynamic redundant reduction algorithms. The dynamic redundant Montgomery reduction (Algorithm 4) requires $\binom{n+2i+1}{2} + n(n+2i+1) = (3n^2 + 3n)/2 + i(4n + 2i + 1)$ base multiplications for a given i. We recall that i bounds the maximum number of multiples of the modulus that can be added. The dynamic redundant Barrett reduction (Algorithm 5)

requires $n^2 + 3n + 1 + i(4n + 2i + 5)$ base multiplications. Table 1 summarizes the complexities.

Table 1: Summary of the complexities of the different modular reduction algorithms in the number of base multiplications for a n-word modulus.

Algorithm	**Number of base multiplications**
Standard Montgomery	$(3n^2 + n)/2$
Dynamic redundant Montgomery	$(3n^2 + 3n)/2 + i(4n + 2i + 1)$
Standard Barrett	$n^2 + 4n + 1$
Dynamic redundant Barrett	$n^2 + 3n + 1 + i(4n + 2i + 5)$

We evaluate a practical implementation of the different reduction algorithms on a AVR 8-bit ATmega 2561 processor [4] running at 16 MHz. A 512-bit modulus is chosen randomly and fixed. Each reduction is implemented in assembly and use the same column-wise multiplication code. The results are summarized in Table 2. Using either Montgomery's or Barrett's technique, we first remark that redundant arithmetic allows time constant reductions as the final subtractions are removed of both standard algorithms. We also note our proposition is more efficient using Barrett reduction. In fact, the dynamic redundant Barrett reduction with $i = 1$ is faster than the standard Barrett. If the randomization is not needed, we can fix the random $k = 0$ in Algorithm 5 and compute a constant time dynamic redundant Barrett with $i = 1$ more efficiently than the classic Barrett. Note that even if the complexity in base multiplication of our algorithm with $i = 1$ is slightly higher than the standard Barrett (see Table 1), our propositions remove the final loop of the Barrett reduction. Hence, for small values of i, we can obtain a dynamic redundant Barrett reduction faster than the standard one.

Table 2: Execution time of each reduction algorithm on a ATmega 2561 processor running at 16 MHz for a fixed modulus of 512 bits and random inputs.

Algorithm	**Time (in ms)**
Standard Montgomery	6.1 or 6.3
Dynamic redundant Montgomery with $i = 1$	8.7
Dynamic redundant Montgomery with $i = 2$	9.3
Standard Barrett	6.4 or 6.6
Dynamic redundant Barrett with $i = 1$	6.3
Dynamic redundant Barrett with $i = 2$	6.6

We define the function $\mathsf{Normalize}(x, m) = x \bmod m$ as a simple loop that computes subtractions $x = x - m$ as long as $x \geq m$. Algorithm 6 illustrates the use of dynamic redundant modular arithmetic in an exponentiation algorithm. We first note that the static redundant arithmetic exponentiation (Algorithm 3) requires one more n-word parameter, *i.e.* we need to store both m and $C = cm$, whereas the dynamic version only needs the modulus m. Moreover, in Algorithm 6, for given pre-computed values β' and R' with a fixed i, one can choose to use a function $\mathsf{rand}()$ that generates random integers in $[0, b^{i'} - 1[$ such that $0 \leq i' \leq i$. In particular, we can fix $i' = 0$ in the exponentiation algorithm if no randomization is needed. In the static redundant case, values C, β_C and R_C need to be pre-computed again if the amount of random needs to be lower for a particular exponentiation.

Algorithm 6 Multiply always exponentiation using dynamic redundant Montgomery arithmetic

Input: positive integers $e = (e_{l-1}, \ldots, e_0)_2, x, m, \beta'$ and R'. Let $\mathsf{rand}()$ be a function that generates a random integer in $[0, b^i - 1[$ for some integer i.
Output: $x^e \bmod m$
1: $X \leftarrow x + \mathsf{rand}()m$
2: $R_0 \leftarrow \mathsf{DRMontRed}(\mathsf{rand}()m, m, R', \beta')$
3: $R_1 \leftarrow \mathsf{DRMontRed}(XR', m, R', \beta')$
4: $i \leftarrow l - 1$, $t \leftarrow 0$
5: **while** $i \geq 0$ **do**
6: $\quad R_0 \leftarrow \mathsf{DRMontRed}(R_0(R_t + \mathsf{rand}()m), m, R', \beta')$
7: $\quad t \leftarrow t \oplus e_i$, $i \leftarrow i - 1 + t$
8: **end while**
9: $R_0 \leftarrow \mathsf{DRMontRed}(R_0 R'^{-1}, m, R', \beta')$
10: $R_0 \leftarrow \mathsf{Normalize}(R_0, m)$
11: **return** R_0

Amiel *et al.* showed in [1] that the Hamming weight of the output of a multiplication $x \times y$ can be distinguished whether $y = x$ or $y \neq x$. Hence, they can defeat atomic implementations of exponentiation algorithms. Using redundant arithmetic as in Algorithm 6, we note that the representation of R_t can be easily randomized prior to the multiplication $R_0 \times R_t$. Hence, even if $t = 0$, the output of the multiplication $R_0 \times R_t$ cannot be distinguished anymore by this attack.

Left-to-right atomic exponentiation algorithms seems particularly vulnerable to combined attacks [3, 16]. In left-to-right algorithms, the value of one register, *e.g.* R_1, is generally fixed to the value of the input message x during the exponentiation. If a precise fault is injected in R_1, an attacker can detect with a simple side-channel analysis when this register is used during the exponentiation, hence the attacker can recover the exponent. Using our dynamic redundant modular propositions, as in the left-to-right Algorithm 6, we can note that the represen-

tation of register R_1 is randomized during the exponentiation. Hence, combined attacks as presented in [3] are no longer able to recover the full exponent.

Dynamic redundant reduction algorithms provide a countermeasure against differential side-channel attacks for public key algorithms. Our solution is an alternative to the classical message blinding [14]. However, as pointed out by Smart *et al.* [18], redundant arithmetic may not be as interesting in elliptic curve cryptography. In fact, if the modulus is a generalized Mersenne prime [19], masking using multiples of the modulus is not as suitable. However, for random modulus, as in RSA, this technique provides a very good defense against differential side-channel attacks for a minimal overhead.

6 Conclusion

We study in this paper the use of redundant arithmetic as a differential side-channel countermeasure. We extend the work of Smart *et al.* [18] by proposing dynamic redundant modular reduction algorithms based on Montgomery's and Barrett's techniques. Our algorithms are parametrized and offer a good flexibility in order to control the amount of randomization as well as the size of the operands. The dynamic randomization of the data inside an exponentiation algorithm can also thwart more refined side-channel attacks. As we remove the final subtraction steps in both standard algorithms, our propositions are time constant and efficient.

Acknowledgments

The authors would like to thank François Dassance, Vincent Verneuil and the anonymous referees of Cardis 2011 for their helpful comments.

References

1. Amiel, F., Feix, B., Tunstall, M., Whelan, C., Marnane, W.: Distinguishing Multiplications from Squaring Operations. Selected Areas in Cryptography, LNCS 5381, 346–360 (2008)
2. Amiel, F., Feix, B., Villegas, K.: Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms. Selected Areas in Cryptography, LNCS 4876, 110–125 (2007)
3. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis. In: FDTC 2007. pp. 92–102. IEEE (2007)
4. ATMEL: ATmega 2561 Data Sheet, `http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf`
5. Barrett, P.: Implementing the Rivest Shamir and Adelman Public Key Encryption Algorithm on a Standard Digital Signal Processor. CRYPTO 1986, LNCS 263, 311–323 (1896)
6. Bosselaers, A., Govaerts, R., Vandewalle, J.: Comparison of Three Modular Reduction Functions. CRYPTO 1993, LNCS 773, 175–186 (1994)

7. Clavier, C., Joye, M.: Universal Exponentiation Algorithm A First Step towards Provable SPA-Resistance. CHES 2001, LNCS 2162, 300–308 (2001)
8. Comba, P.: Exponentiation Cryptosystems on the IBM PC. IBM Syst. J. 29, 526–538 (1990)
9. Dhem, J.F.: Design of an efficient public-key cryptographic library for RISC-based smart cards. Ph.D. thesis, Université Catholique de Louvain (1998)
10. Golić, J., Tymen, C.: Multiplicative Masking and Power Analysis of AES. CHES 2002, LNCS 2535, 31–47 (2002)
11. Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. CRYPTO 1996, LNCS 1109, 104–113 (1996)
12. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. CRYPTO 1999, LNCS 1666, 388–397 (1999)
13. Menezes, A.J., Vanstone, S.A., Van Oorschot, P.C.: Handbook of Applied Cryptography. CRC Press, Inc. (1996)
14. Messerges, T., Dabbish, E., Sloan, R.: Power Analysis Attacks of Modular Exponentiation in Smartcards. CHES 1999, LNCS 1717, 724–724 (1999)
15. Montgomery, P.: Modular Multiplication Without Trial Division. Mathematics of computation 44(170), 519–521 (1985)
16. Schmidt, J., Tunstall, M., Avanzi, R., Kizhvatov, I., Kasper, T., Oswald, D.: Combined Implementation Attack Resistant Exponentiation. LATINCRYPT 2010, LNCS 6212, 305–322 (2010)
17. Scott, M., Szczechowiak, P.: Optimizing Multiprecision Multiplication for Public Key Cryptography. Cryptology ePrint Archive, Report 2007/299 (2007)
18. Smart, N., Oswald, E., Page, D.: Randomised Representations. Information Security, IET 2(2), 19–27 (2008)
19. Solinas, J.: Generalized Mersenne Numbers. Technical report (1999)
20. Walter, C.: Montgomery Exponentiation Needs no Final Subtractions. Electronics letters 35(21), 1831–1832 (2002)