A Protected Division Algorithm

[Published in P. Honeyman, Ed., *Fifth Smart Card Research and Advanced Application Conference (CARDIS '02)*, pp. 69–74, Usenix Association, 2002.]

Marc Joye and Karine Villegas

Gemplus Card International, Card Security Group
La Vigie, Avenue des Jujubiers, ZI Athélia IV, 13705 La Ciotat Cedex, France
{marc.joye, karine.villegas}@gemplus.com
http://www.geocities.com/MarcJoye/ – http://www.gemplus.com/smart/

Abstract. Side-channel analysis is a powerful tool for retrieving secrets embedded in cryptographic devices such as smart cards. Although several practical solutions have been proposed to prevent the leakage of sensitive data, mainly the protection of the basic cryptographic operation itself has been thoroughly investigated. For example, for exponentiation-based cryptosystems (including RSA, DH or DSA), various exponentiation algorithms protected against side-channel analysis are known. However, the exponentiation algorithm itself or the underlying crypto-algorithm often involve division operations (for computing a quotient or a remainder). The first case appears in the normalization (resp. denormalization) process in fast exponentiation algorithms and the second case appears in the data processing before (resp. after) the call to the exponentiation operation.

This paper proposes an efficient division algorithm protected against simple side-channel analysis. The proposed algorithm applies equally well to software and hardware implementations. Furthermore, it does not impact the running time nor the memory requirements.

Keywords. Division algorithms, smart cards, side-channel analysis, SPA protected implementations.

1 Introduction

Significant progress has been made these last years to secure cryptographic devices (e.g., smart cards) against side-channel analysis. Side-channel analysis [2, 3] is a clever technique exploiting side-channel information (e.g., power consumption) to retrieve secret information involved in the execution of a carelessly implemented crypto-algorithm. The threat is now clearly understood by implementors and various countermeasures have been suggested.

The basic operation underlying most public-key crypto-algorithms is the modular exponentiation. To name a few, this includes the RSA cryptosystem, the Diffie-Hellman key exchange or the DSA signature scheme. The resistance of modular exponentiation with respect to side-channel analysis is discussed in many papers (e.g., see [4] where both attacks and counter-measures are presented). A far less studied operation is that of division: to the authors' best knowledge, there is no paper in the public literature addressing this issue. This is most unfortunate as nearly all implementations of exponentiation-based cryptosystems use the division operation as well.

Several specialized modular multiplication algorithms (and therefore the corresponding modular exponentiation algorithms) require a normalization step involving an integer division. Typical examples include Barrett algorithm [5] or Quisquater algorithm [6] (see also [7]). For computing $a \cdot b \bmod m$, these two algorithms take on input a normalization factor of the form $\mu = \lfloor 2^t/m \rfloor$. If the division algorithm used for evaluating μ is prone to side-channel analysis then the value of m (or some related information) can be recovered. When m is a secret data, this compromises the security of the cryptosystem. For example, this occurs when RSA decryption (or signature) is speeded up through Chinese remaindering [8] because then modulus m is successively one of the two secret RSA primes, p_1 and p_2. A second example of division algorithm manipulating secret data is when RSA is used with Chinese remaindering and operand x in the computation of $x^d \bmod \{p_1, p_2\}$ is first explicitly reduced modulo p_i prior to the exponentiation $x^d \bmod p_i$, for $i = 1, 2$.

An algorithm commonly used for computing integer divisions is the classical binary pencil-and-paper method (or a variant thereof). This algorithm presents the advantage of requiring no extra memory requirements. However, as we will see, it may yield the value of quotient $q = a \operatorname{div} b$ during its computation by simple side-channel analysis. This paper is aimed at transforming this algorithm into a division algorithm protected against simple side-channel analysis while preserving the efficiency (memory-wise) of the classical algorithm. Actually, the resulting algorithm will not only be protected against simple side-channel analysis but will further be faster than the classical algorithm, with the same memory requirements. As a result, we obtain a protected division algorithm that is few greedy in memory and is particularly suited to a hardware implementation or to a software implementation in a constrained environment like a smart card.

The rest of this paper is organized as follows. The next section reviews the classical binary pencil-and-paper division algorithm. Its security towards simple side-channel analysis is studied in Section 3. Building on the pencil-and-paper method, we then propose in Section 4 our protected yet more efficient division algorithm. Finally, we conclude in Section 5.

Disclaimer. This paper only addresses security against *simple* side-channel analysis, that is, side-channel analysis from a *single* measurement of certain side-channel information. In particular, it is not concerned with differential analysis (such as DPA) or more sophisticated methods.

2 Pencil-and-Paper Division Method

Given a and b on input, the binary pencil-and-paper algorithm evaluates the quotient $q = a \operatorname{div} b$ (alternatively, we use the notation $q = \lfloor \frac{a}{b} \rfloor$) and the remainder $r = a \bmod b$. The binary representations of a and b are respectively given by $a = (a_{m-1}, \ldots, a_0)_2$ and $b = (b_{n-1}, \ldots, b_0)_2$ with $b_{n-1} \neq 0$.

It is easy to see that the pencil-and-paper division of two integers amounts to the simpler problem of dividing a $(n+1)$-bit integer A by a n-bit integer b and then to re-iterate the process [1]. We must have $0 \leq A/b < 2$, which is satisfied whenever $b_{n-1} \neq 0$ (see above restriction).

Since we are working in basis 2, the two possible values for the quotient bit $\lfloor A/b \rfloor$ are 0 or 1. So we subtract b from A and test whether the obtained result is nonnegative; if so then $\lfloor A/b \rfloor = 1$, and $\lfloor A/b \rfloor = 0$ otherwise. Remark that $\lfloor A/b \rfloor = 1$ if and only if $A - b \geq 0$.

The length of $r = a \bmod b$ plus the length of $q = a \operatorname{div} b$ is smaller than or equal to $(m+1)$ bits. Indeed, the length of r is at most n bits since $r < b$; and the length of q is at most $(m-n+1)$ bits since $q = \lfloor a/b \rfloor \leq \lfloor a/(b_{n-1}2^{n-1}) \rfloor = a \operatorname{div} 2^{n-1} = (a_{m-1}, \ldots, a_{n-1})_2$, a $(m-n+1)$-bit value. In order to save memory, the quotient and remainder will be written in the register containing a (augmented with one leading bit).

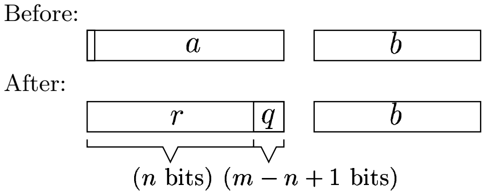

Fig. 1. Memory configuration.

It is useful to introduce some notations. For a k-bit integer a, we denote by $\texttt{SHL}_k(a,1)$ the operation consisting in shifting a of one bit to the left; the outgoing bit is affected to the `carry`. For two k-bit integers a and b, we write $\texttt{ADD}_k(a,b)$ for the addition of a and b; variable `carry` is set to 1 if there is a carry in the addition and `carry` is set to 0 otherwise. Remark that $\texttt{SHL}_k(a,1)$ can equivalently be obtained as $\texttt{ADD}_k(a,a)$. There is usually no subtraction operation available for large integers. The subtraction of b from a is obtained by first computing the two's complement of b, denoted by $\bar{b}$, and then by adding $\bar{b}$ to a. Indeed, if b is a k-bit integer then $b + \bar{b} = 2^k$ and so $a - b = a + \bar{b} \pmod{2^k}$. We write $\texttt{CPL2}_k(a)$ the operation consisting in taking the two's complement of a k-bit integer a. Symbols $\vee$, $\wedge$ and $\oplus$ refer to the bit-wise logical operations OR, AND and XOR, respectively. For a bit σ, the negation of σ (i.e., its complementary value) is denoted by $\neg\sigma$. Finally, the notation $\operatorname{lsb}(a)$ refers to the least significant bit of an integer a.

We can now present the classical binary pencil-and-paper division algorithm. On input a and b, this algorithm computes both the values of $a \operatorname{div} b$ and of $a \bmod b$. In order to work, a is artificially augmented with one bit (initially set to 0) at the most significant position. Moreover, to ease the exposition, variable A represents the n most significant bits of a, i.e., $A = (0, a_{m-1}, \ldots, a_{m-n+1})$.

```
Input:  $a = (0, a_{m-1}, \ldots, a_0)_2$ and $b = (b_{n-1}, \ldots, b_0)_2$
Output: $q = a \operatorname{div} b$ and $r = a \bmod b$
$b \leftarrow$ CPL2$_n(b)$ /* $b$ = "$-b$" */
for $j = 1$ to $(m - n + 1)$ do
    $a \leftarrow$ SHL$_{m+1}(a, 1)$; $\sigma \leftarrow$ carry /* Shift */
    $A \leftarrow$ ADD$_n(A, b)$; $\sigma \leftarrow \sigma \vee$ carry /* Subtract */
    if $(\neg\sigma)$ then /* Correction */
            $b \leftarrow$ CPL2$_n(b)$; $A \leftarrow$ ADD$_n(A, b)$; $b \leftarrow$ CPL2$_n(b)$
        else lsb$(a) = 1$
endfor
$b \leftarrow$ CPL2$_n(b)$
```

Fig. 2. Binary pencil-and-paper division algorithm.

Remainder r is in A followed by the quotient q. The correctness of the algorithm follows by observing that if $a \leftarrow$ SHL$_{m+1}(a, 1)$ generates a carry then $a_m = 1$ (before shifting) and so b must be subtracted; moreover if $a_m = 0$ (before shifting) and $A \leftarrow$ ADD$_n(A, b)$ generates a carry (i.e., $A - b \geq 0$ before the subtraction) then again b must be subtracted.

Example 1. Suppose that we want to compute $a \operatorname{div} b$ and/or $a \bmod b$ with $a = 4096 = (1000000000000)_2$ and $b = 81 = (1010001)_2$. The 2's complement of b is $\bar{b} = (0101111)_2$. We obtain $r = 46 = (101110)_2$ and $q = 50 = (110010)_2$.

	a														σ
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	$\star$
Shift	1	0	0	0	0	0	0	0	0	0	0	0	0	$\underline{0}$	0
Subtract	1	1	0	1	1	1	1								0
Correction	1	0	0	0	0	0	0								
Shift	0	0	0	0	0	0	0	0	0	0	0	0	$\underline{0}$	0	1
Subtract	0	1	0	1	1	1	1							1	1
Shift	1	0	1	1	1	1	0	0	0	0	0	$\underline{0}$	1	0	0
Subtract	0	0	0	1	1	0	1							1	1
Shift	0	0	1	1	0	1	0	0	0	0	$\underline{0}$	1	1	0	0
Subtract	1	0	0	1	0	0	1								0
Correction	0	0	1	1	0	1	0								
Shift	0	1	1	0	1	0	0	0	0	$\underline{0}$	1	1	0	0	0
Subtract	1	1	0	0	0	1	1								0
Correction	0	1	1	0	1	0	0								
Shift	1	1	0	1	0	0	0	0	$\underline{0}$	1	1	0	0	0	0
Subtract	0	0	1	0	1	1	1							1	1
Shift	0	1	0	1	1	1	0	$\underline{0}$	1	1	0	0	1	0	0
Subtract	1	0	1	1	1	0	1								0
Correction	0	1	0	1	1	1	0	$\underline{0}$	1	1	0	0	1	0	

3 Security Analysis

This section explains why security may be a concern when implementing a division algorithm.

Back to the binary pencil-and-paper division algorithm (Fig. 2), we see that the quotient is constructed bit by bit. According to its value, each quotient bit is obtained by a different sequence of operations. At step j in the `for`-loop, if the obtained quotient bit, say q_j, has value 0 then the following operations were performed:

- a shift [$\texttt{SHL}_{m+1}(a, 1)$]
- an addition [$\texttt{ADD}_n(A, b)$]
- a "correction" [$\texttt{CPL2}_n(b)$; $\texttt{ADD}_n(A, b)$; $\texttt{CPL2}_n(b)$]

along with some logical operations; whereas if $q_j = 1$ then the following two operations were performed:

- a shift [$\texttt{SHL}_{m+1}(a, 1)$]
- an addition [$\texttt{ADD}_n(A, b)$]

along with some logical operations. If it is possible to distinguish these two sequence of operations during the course of the algorithm then the value of quotient bit q_j (and thus of the whole quotient q) can be recovered. Such a means may be provided by monitoring the power consumption (i.e., the side-channel is the power consumption). The next figure represents a power trace resulting from the execution of the pencil-and-paper division algorithm on a chip equipped with a crypto-coprocessor. The operands are large numbers and the various operations (shift, addition or two's complement) are performed by the crypto-coprocessor.

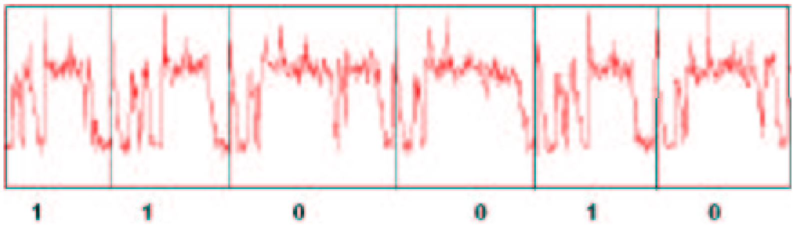

Fig. 3. A power trace of the pencil-and-paper division algorithm.

We can identify two different patterns in Fig. 3: one corresponds to the case $q_j = 0$ (i.e., a longer pattern involving an additional "correction") and the other one corresponds to the case $q_j = 1$. Therefore, the value of quotient $q = a \operatorname{div} b$ can easily read from the power trace.

If quotient q (or related data) is secret, this above implementation is not secure. Consider the example of RSA implemented with Chinese remaindering. Let

$N = p_1 p_2$ be an RSA modulus (the values of p_1 and p_2 are secret). The computation of $y = x^d \bmod N$ is carried out as $y = \mathrm{CRT}(y_1, y_2)$ where $y_i = {x_i}^d \bmod p_i$ with $x_i = x \bmod p_i$, for $i = 1, 2$. Suppose that x_1 and x_2 are computed by the binary pencil-and-paper algorithm as given in Fig. 2. Then, by simple power analysis[1] (SPA), the values of $q_1 := \lfloor x/p_1 \rfloor$ and $q_2 := \lfloor x/p_2 \rfloor$ can be recovered from the corresponding power traces. Suppose further that $x = N - r$ for some $0 < r < p_1$. Then

$$q_1 = \left\lfloor \frac{N-r}{p_1} \right\rfloor = \left\lfloor p_2 - \frac{r}{p_1} \right\rfloor = p_2 - 1$$

and so the secret RSA primes are given by $p_2 = q_1 + 1$ and $p_1 = N/p_2$.

4 A Protected Method

The previous analysis illustrates that non-constant code may reveal sensitive data, thereby compromising the security of the cryptosystem. A first idea to make the code constant consists in adding some dummy operations and in making implicit the `if-then-else` statement. Such a solution is however unsatisfactory as dummy operations penalize the running time. This is especially true when the dummy operations are time-consuming

Rather, we exploit the following observation. At each iteration, the pencil-and-paper algorithm (Fig. 2) computes $a \leftarrow 2a - b\,2^{m-n+1}$. In the case of an unsuccessful guess (i.e., when $\sigma = 0$), one has to restore the value of a by setting $A \leftarrow A + b$. This restoring step can be avoided by noting that $2(a + b\,2^\alpha) - b\,2^\alpha = 2a + b\,2^\alpha$. We then obtain the non-restoring variant of the classical binary pencil-and-paper division algorithm (see Fig. 4). An additional variable, σ', keeps track of the value of bit σ in the previous iteration. Bit β keeps track of the 'sign' of b.

The algorithm given in Fig. 4 does not behave regularly and so may also be subject to side-channel analysis. According to the values of β and of σ', register b is unchanged or replaced by its two's complement. A first step towards side-channel protection consists in always performing a two's complement followed by an addition, whatever the values of β and σ'. To this end, when register b needs not to be replaced by its two's complement, a dummy two's complement —on a register, say register c, that does not impact the computation— is executed. We call d_{addr} the address of the register containing the value that will be replaced by its two's complement (d_{addr} will be b_{addr} or c_{addr}).

It is also worth noting that σ can be updated as $\sigma \leftarrow \sigma'(\sigma \vee \texttt{carry}) + (\neg\sigma')(\sigma \wedge \texttt{carry})$, which can equivalently be rewritten as

$$\sigma \leftarrow (\sigma \wedge \sigma') \oplus (\sigma \wedge \texttt{carry}) \oplus (\sigma' \wedge \texttt{carry}) \enspace .$$

Finally, noting that the shift operation sets the least significant bit of a to 0, the line `[if (`σ`) then` $\mathrm{lsb}(a) = 1$`]` can be replaced by $\mathrm{lsb}(a) \leftarrow \sigma$.

[1] That is, a side-channel analysis using a single power consumption measurement as side-channel information.

We use β and γ variables to keep track of the 'sign' of the value contained in the registers located at b_{addr} and c_{addr}, respectively. The convention is $\beta = 0$ (resp. $\gamma = 0$) when the value located at b_{addr} (resp. c_{addr}) is the original value, and $\beta = 1$ (resp. $\gamma = 1$) when the value located at b_{addr} (resp. c_{addr}) is the two's complement of the original value. We have the following truth table:

σ'	β	γ	β	γ
0	0	0	1	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

where-from we derive the outgoing values $\beta \leftarrow \neg\sigma'$ and $\gamma \leftarrow \gamma \oplus \sigma' \oplus \beta$.

Putting all together, we finally obtain the algorithm given in Fig. 4. One may argue that our algorithm is not code-constant because of the three last `if-then` statements. We note however that the two first are not mandatory to make the algorithm working but are merely performed to reset the registers containing b and c to their original values. Finally, the last `if-then` only reveals the least significant bit of the quotient; when this is a secret value a dummy operation can be applied to mask the potential addition $\mathtt{ADD}_n(A, b)$.

Example 2. We take the same example as before: $a = 4096 = (1000000000000)_2$ and $b = 81 = (1010001)_2$ ($\bar{b} = 0101111$). As detailed below, we obtain $r = 46 = (101110)_2$ and $q = 50 = (110010)_2$.

								a							σ	β
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	$\star$	1
Shift	1	0	0	0	0	0	0	0	0	0	0	0	0	$\underline{0}$	0	1
$\mathtt{CPL2}_n(b)$ & $\mathtt{ADD}_n(A, b)$	1	1	0	1	1	1	1								0	0
Shift	1	0	1	1	1	1	0	0	0	0	0	0	$\underline{0}$	0	1	0
$\mathtt{CPL2}_n(b)$ & $\mathtt{ADD}_n(A, b)$	0	1	0	1	1	1	1							1	1	1
Shift	1	0	1	1	1	1	0	0	0	0	0	$\underline{0}$	1	0	0	1
$\mathtt{CPL2}_n(b)$ & $\mathtt{ADD}_n(A, b)$	0	0	0	1	1	0	1							1	1	0
Shift	0	0	1	1	0	1	0	0	0	0	$\underline{0}$	1	1	0	0	0
$\mathtt{CPL2}_n(c)$ & $\mathtt{ADD}_n(A, b)$	1	0	0	1	0	0	1								0	0
Shift	0	0	1	0	0	1	0	0	0	$\underline{0}$	1	1	0	0	1	0
$\mathtt{CPL2}_n(b)$ & $\mathtt{ADD}_n(A, b)$	1	1	0	0	0	1	1								0	1
Shift	1	0	0	0	1	1	0	0	$\underline{0}$	1	1	0	0	0	1	1
$\mathtt{CPL2}_n(c)$ & $\mathtt{ADD}_n(A, b)$	0	0	1	0	1	1	1							1	1	1
Shift	0	1	0	1	1	1	0	$\underline{0}$	1	1	0	0	1	0	0	1
$\mathtt{CPL2}_n(b)$ & $\mathtt{ADD}_n(A, b)$	1	0	1	1	1	0	1								0	0
$\mathtt{CPL2}_n(b)$	(Final corr. on b)															
Final corr.	0	1	0	1	1	1	0	$\underline{0}$	1	1	0	0	1	0	0	0

```
Input:  $a = (a_{m-1}, \ldots, a_0)_2$ and $b = (b_{n-1}, \ldots, b_0)_2$
Output: $q = a \operatorname{div} b$ and $r = a \bmod b$
---------------------------------------------------------------
$\sigma' \leftarrow 1$; $\beta \leftarrow 1$
for $j = 1$ to $(m - n + 1)$ do
   $a \leftarrow \mathtt{SHL}_{m+1}(a, 1)$; $\sigma \leftarrow$ carry
   if $(\sigma')$ then if $(\beta)$ then $b \leftarrow \mathtt{CPL2}_n(b)$; $\beta \leftarrow 0$
                 $A \leftarrow \mathtt{ADD}_n(A, b)$; $\sigma \leftarrow \sigma \vee$ carry
            else if $(\neg\beta)$ then $b \leftarrow \mathtt{CPL2}_n(b)$; $\beta \leftarrow 1$
                 $A \leftarrow \mathtt{ADD}_n(A, b)$; $\sigma \leftarrow \sigma \wedge$ carry
   if $(\sigma)$ then $\mathrm{lsb}(a) = 1$
   $\sigma' \leftarrow \sigma$
endfor
if $(\neg\beta)$ then $b \leftarrow \mathtt{CPL2}_n(b)$
if $(\neg\sigma)$ then $A \leftarrow \mathtt{ADD}_n(A, b)$
```

Fig. 4. Non-restoring binary division algorithm.[2]

```
Input:  $a = (a_{m-1}, \ldots, a_0)_2$ and $b = (b_{n-1}, \ldots, b_0)_2$
Output: $q = a \operatorname{div} b$ and $r = a \bmod b$
---------------------------------------------------------------
$\sigma' \leftarrow 1$; $\beta \leftarrow 1$; $\gamma \leftarrow 1$
for $j = 1$ to $(m - n + 1)$ do
   $a \leftarrow \mathtt{SHL}_{m+1}(a, 1)$ /*Shift*/
   $\sigma \leftarrow$ carry; $\delta \leftarrow \sigma' \oplus \beta$
   $d_{addr} \leftarrow b_{addr} + \delta(c_{addr} - b_{addr})$
   $d \leftarrow \mathtt{CPL2}_n(d)$ /*Two's complement*/
   $A \leftarrow \mathtt{ADD}_n(A, b)$ /*Addition*/
   $\sigma \leftarrow (\sigma \wedge \sigma') \oplus (\sigma \wedge \mathtt{carry}) \oplus (\sigma' \wedge \mathtt{carry})$
   $\beta \leftarrow \neg\sigma'$; $\gamma \leftarrow \gamma \oplus \delta$; $\sigma' \leftarrow \sigma$
   $\mathrm{lsb}(a) = \sigma$
endfor
/*Final correction*/
if $(\neg\beta)$ then $b \leftarrow \mathtt{CPL2}_n(b)$
if $(\neg\gamma)$ then $c \leftarrow \mathtt{CPL2}_n(c)$
if $(\neg\sigma)$ then $A \leftarrow \mathtt{ADD}_n(A, b)$
```

Fig. 5. Our protected division algorithm.[2]

[2] Again variable A represents the n most significant bits of variable a, i.e., $A = (0, a_{m-1}, \ldots, a_{m-n+1})$.

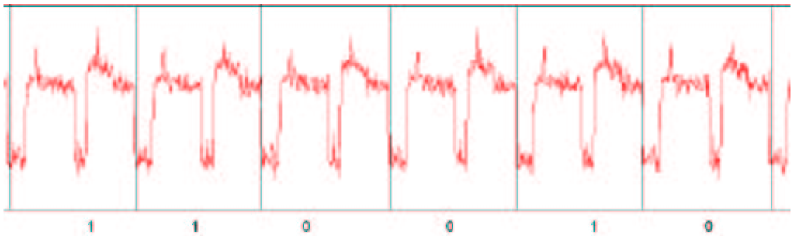

Fig. 6. A power trace of our division algorithm.

The corresponding power trace is given in Fig. 6. Remark that the power trace is now the repetition of a same pattern, regardless the value of the quotient bit.

Finally, as a side-effect, it is easy to see that, on average, our protected algorithm outperforms the classical pencil-and-paper method, with the same memory constraints (cf. Fig. 1).

Table 1. Comparison with the classical method.

	$\mathtt{ADD}_n$	$\mathtt{SHL}_{m+1}$	$\mathtt{CPL2}_n$
Classical	$\frac{3}{2}(m-n+1)$	$m-n+1$	$m-n+3$
Protected	$m-n+\frac{3}{2}$	$m-n+1$	$m-n+2$

5 Conclusions

This paper presented a new division algorithm preventing simple side-channel analysis. The proposed algorithm is well suited to a hardware implementation or to a software implementation in a constrained environment. Remarkably, it does not require additional resources (time or memory) and is even faster than the classical binary method.

Obviously, we note that SPA-like analysis highly depends on the hardware and special care must be paid by the implementor. In this respect, the proposed method can be seen as a useful framework for designing protected and, as shown in the paper, efficient division algorithms.

References

1. D.E. Knuth, *The Art of Computer Programming – Seminumerical Algorithms*, Addison-Wesley, 2000.
2. P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology – CRYPTO '96*, vol. 1109 of *Lecture Notes in Computer Science*, pp. 104–113, Springer-Verlag, 1996.

3. P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Advances in Cryptology – CRYPTO ’99*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 388–397, Springer-Verlag, 1999.
4. T.S. Messerges, E.A. Dabbish, and R.H. Sloan, “Power analysis attacks of modular exponentiation in smartcards,” in *Cryptographic Hardware and Embedded Systems (CHES ’99)*, vol. 1717 of *Lecture Notes in Computer Science*, pp. 144–157, Springer-Verlag, 1999.
5. P. Barrett, “Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processing,” in *Advances in Cryptology – CRYPTO ’86*, vol. 263 of *Lecture Notes in Computer Science*, pp. 311–323, Springer-Verlag, 1987.
6. J.-J. Quisquater, “Encoding system according to the so-called RSA method, by means of a microcontroller and arrangement implementing this system.” U.S. patent #5,166,978, Nov. 1992.
7. A. Bosselaers, R. Govaerts, and J. Vandewalle, “Comparison of three modular reduction functions,” in *Advances in Cryptology – CRYPTO ’93*, vol. 773 of *Lecture Notes in Computer Science*, pp. 175–186, Springer-Verlag, 1994.
8. J.-J. Quisquater and C. Couvreur, “Fast decipherment algorithm for RSA public-key cryptosystem,” *Electronics Letters*, vol. 18, pp. 905–907, Oct. 1982.