GF(2^m) Finite-Field Multipliers with Reduced Activity Variations

Danuta Pamula[1,2] and Arnaud Tisserand[1]

[1] IRISA, CNRS, INRIA, Univ. Rennes 1, Lannion, France.
arnaud.tisserand@irisa.fr
[2] Silesian University of Technology, Gliwice, Poland
danuta.pamula@polsl.pl

Abstract. Electrical activity variations in a circuit are one of the information leakage used in side channel attacks. In this work, we present GF(2^m) multipliers with reduced activity variations for asymmetric cryptography. Useful activity of typical multiplication algorithms is evaluated. The results show strong shapes, which can be used as a small source of information leakage. We propose modified multiplication algorithms and multiplier architectures to reduce useful activity variations during an operation.

Keywords: finite field arithmetic, cryptography, side channel attacks

1 Introduction

Side channel attacks (SCA) are nowadays a major threat for embedded cryptographic systems. Power analysis based SCAs exploit correlations between internal secret values (e.g. keys) and the current, which flows in the circuit [14]. Similar information leakage can be exploited from electromagnetic radiations [7].

In elliptic curve cryptography (ECC) [9], many SCAs [19] have been proposed. To protect circuits against those attacks researchers propose various countermeasures, or protections, for ECC see [10]. Moreover, specific protections at the arithmetic level have been proposed for ECC (and other asymmetric cryptosystems). For instance, addition chains allow performing only one type of operation, point addition, during scalar multiplications [2]. In [3] randomized and very redundant representations of the scalar are used. But these protections are at the curve level not the finite field one. Efficient and secure computation units for finite-field arithmetic are important elements of ECC processors. For instance, see [18] and [8] as examples of GF(2^m) ECC processors.

In this work, we investigate protection elements at the field level in GF(2^m) multiplication algorithms and their architectures for ECC. The proposed solutions are not autonomous countermeasures but an additional protection element, which should enhance higher-level countermeasures. In some SCAs, power variations provide sources of information leakage about the executed operations.

Instantaneous power is linked with the number of useful transitions in the operator. Useful transitions are the theoretical changes during the operation

F. Özbudak and F. Rodríguez-Henríquez (Eds.): WAIFI 2012, LNCS 7369, pp. 152–167, 2012.

(from one clock cycle to the next one). This is also called useful circuit activity. To estimate information leakage in typical GF(2^m) multipliers, we first accurately measured their useful activity. The results show very strong shapes for useful activity variations. These shapes reveal multiplications time boundaries. Next, we propose modifications of the multiplication algorithms and architectures for reducing useful activity variations during the computation.

The paper presents background on power consumption in digital circuits and activity evaluation methods in Section 2. In Section 3, GF(2^m) multiplication algorithms used for our experiments are described. In Section 4, we present the results and an analysis of useful activity measurements as an estimation of information leakage source. Section 5 presents our modifications for reducing the activity variations in the multipliers and the new implementation results.

2 Activity in Hardware Arithmetic Operators

This section gives a short introduction to electrical activity in digital integrated circuits and its links to SCAs. Power analysis based SCAs use possible correlations between internal secret values (e.g. keys) and information leakage related to instantaneous power of the executed operations (see [14] for details).

Instantaneous power at time t is $P_{\mathrm{DD}}(t) = i_{\mathrm{DD}}(t) \times V_{\mathrm{DD}}$ where $i_{\mathrm{DD}}(t)$ is the instantaneous current and V_{DD} is the power supply. Power consumption components are: *static* power and *dynamic* power. See [26, Sec. 4.4] for circuit-level details in CMOS circuits. Static power does not depend on circuit activity and is not used in this work. Dynamic power is due to circuit activity: charging and discharging load/parasitic capacitances and short-circuit currents. It strongly depends on the executed operations and data values. Dynamic power variations are used as a source of information leakage for power attacks.

Dynamic power components are: *useful* activity and *parasitic* activity as illustrated on Figure 1. Useful (or theoretical) activity is due to complete and stable transitions required by computations from one clock cycle to the next one (i.e. $0 \rightarrow 1$ and $1 \rightarrow 0$ for each bit). Parasitic (or glitching) activity is due to non-useful transitions. For instance, in case of non-equal arrival times for a gate inputs, the output may have multiple transitions before reaching a steady state.

Parasitic activity in GF(2^m) multipliers is small. This is not the case for all arithmetic operators (e.g. operators in high-performance CPUs [24]). In GF(2^m) arithmetic units, the logical depth is small. Power consumption of memory elements (e.g. flip-flops) used in GF(2^m) multipliers is important compared to

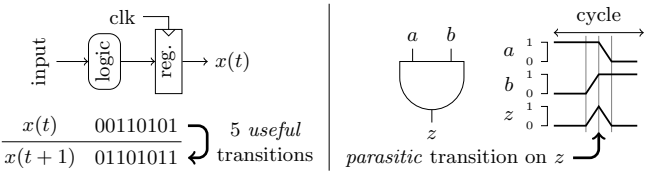

Fig. 1. Useful (left) and parasitic (right) transitions

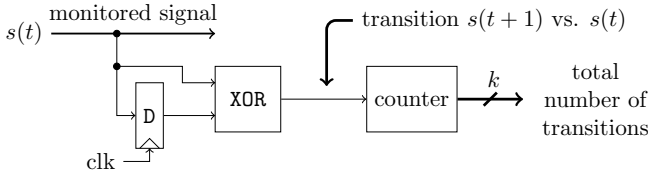

Fig. 2. Activity counter architecture for a 1-bit signal $s(t)$ (control not represented)

power consumption in logic gates. In this work, we only focus on useful activity as a large contribution to $i_{\mathrm{DD}}(t)$.

Several methods can be used to evaluate useful activity: cycle accurate and bit accurate (CABA) simulation of a low-level architecture description, electrical simulation or FPGA emulation. Fast high-level behavioral simulation is not sufficient to catch cycle accurate and bit-level coding aspects. As the target operators have large operands (e.g. 160 to 500 bits for ECC) and long computations CABA simulation would be to slow. This is even more critical with electrical simulation. So, we use FPGA emulation for evaluating useful activity. An activity counter is added on each monitored signal [25]. It counts the number of useful transitions as illustrated on Figure 2. The `D` flip-flop and the `XOR` gate produce a `1` for each useful transition between $s(t+1)$ and $s(t)$. The k-bit counter accumulates transitions numbers (k depends on test vector length).

We insert activity counters at the output of each internal register and for each bit of the multiplier. Comparisons with electrical simulations in [25] show that this is reasonable assumption for small parasitic activity. Outputs of all `XOR` gates (radix-1 representation of transitions number) are compressed into a binary value as the total transitions number for cycle t. This value is stored in a memory as an estimation of $i_{\mathrm{DD}}(t)$. At the end of the evaluation period, the memory content is sent to the host computer for the analysis step. Using FPGA emulation, it is possible to quickly and accurately evaluate useful activity in GF(2^m) finite-field multipliers for large and relevant test vectors (this cannot be done using "slow" software simulations). Activity counters do not change the multiplier mathematical behavior. FPGA emulation is obviously CABA.

In Section 4, we present the implementation results of GF(2^m) multipliers without and with activity counters. Table 1 reports huge area overhead and about a $\div 3$ frequency decrease due to the counters. These overheads are very important, but they only appear during evaluation not in final circuit. FPGA emulation leads to activity evaluation running at more than 100 MHz (cf. Table 1) which would not be possible using software simulations.

3 GF(2^m) Finite-Field Multiplication Operators

A complete introduction to finite fields and their applications can be found in book [13]. Algorithms for finite-field arithmetic are presented in book [4]. Binary-field extension algorithms are presented in [21, chapter 6]. A summary on GF(2^m) multiplication algorithms is given in paper [22].

Our goal was to design efficient finite-field multipliers in such a way that their architectures can be easily modified to add protection against SCAs. We have analyzed many algorithms, with different variants, to be able to take and combine those parts, which will allow us to create the most efficient algorithm fulfilling, assumed requirements. We have considered GF(2^m) elements represented by polynomial basis of the form $\{1, x, x^2, ..., x^{m-2}, x^{m-1}\}$. As a result of our study, we have prepared three GF(2^m) multipliers based on three different algorithms: classical two-step, Montgomery and Mastrovito.

All proposed hardware solutions are analyzed for standard size $m = 233$ (similar results are obtained for other field extension sizes). See [9, annexes A and B] for ECC standards and parameters. The multipliers are described using VHDL, synthesized, place and routed using Xilinx ISE 12.2 environment and implemented in a Xilinx Virtex-6 LX240T FPGA.

3.1 Classic Two-Step Multiplication Algorithm

Classic multiplication comprises two steps: multiplication $d(x) = a(x)b(x)$ and reduction $c(x) = d(x) \bmod f(x)$. There exist many versions of two-step multipliers, which combine different methods for multiplication and reduction. In our hardware solution we have combined Karatsuba-Ofman multiplication principle with features of matrix-vector approach. Moreover to perform reduction we have used classical method optimized for a specific irreducible polynomial.

Multiplication part: In order to avoid managing large vectors a, b we have partitioned them into smaller vectors according to divide-and-conquer method optimized with Karatsuba-Ofman trick [11]. Karatsuba-Ofman partitioning of polynomials is assumed for polynomials of even sizes. Because field sizes used for cryptographic purposes are usually odd, we add redundant zeroes on missing most significant positions of A_H, B_H chunks. Utilizing Karatsuba-Ofman optimization aiming at reducing number of the most complex operation in the equation, we denote multiplication as follows:

$$\begin{aligned} d(x) = a(x)b(x) &= (x^{\frac{m}{2}}A_H + A_L)(x^{\frac{m}{2}}B_H + B_L) \\ &= x^m A_H B_H + x^{\frac{m}{2}}((A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L) + A_L B_L. \end{aligned}$$

After vast analysis we have found that the best results in terms of speed and area, for most input sizes are for halved inputs. For some partitions sizes, designed multipliers were significantly faster but also significantly bigger and otherwise. We had to find some trade-off. Thus having $m = 233$ and utilizing Karatsuba-Ofman trick to perform multiplication we have decided to use three 117-bit multipliers.

Our 117-bit multipliers are based on matrix-vector approach. There, polynomial $a(x)$ is represented by a specific matrix A of size $2m-1 \times m$, in which each column represents consecutive left shifts of $a(x)$, element $b(x)$ is represented in form of m-bit vector and product $d(x)$ is also a vector but of size $(2m-1)$.

Our idea for a multiplier based on this approach is to store only two columns of the matrix A at a time, which in fact means working on two registers representing matrix A columns, exchanging in the first values of consecutive columns of A and accumulating partial results in the other. We may actually regard one vector as a column of matrix A and the other as a vector storing product d.

Reduction part: Reduction module uses reduction method optimized for irreducible polynomial $f(x)$. In classical method (see [9], [4]) one looks for bits equal 1 in the upper part of $d(x)$, from ranks $(2m-2)$ down to m, and step by step reduces vector $d(x)$, XORs vector $d(x)$ by appropriate shift left of irreducible polynomial $f(x)$. The drawback of this method is that it is very time consuming. Observing properties of special irreducible polynomials (e.g. trinomials, pentanomials) one may optimize classical algorithm. In case of field size $m = 233$, the irreducible trinomial $f(x) = x^{233} + x^{74} + 1$ is used. With such polynomial we were able to significantly decrease number of steps needed to reduce the product as depicted in Figure 3. Similar algorithms can be found in [9].

CONSTANTS: $f(x) = x^{233} + x^{74} + 1$,	
INPUT: $d(x)$,	
OUTPUT: $c(x) = d(x) \bmod f(x)$	
1. $e = d[2m-2, ..., m]$	// assign part of vector d to e
2. $e1 = e \times f$	
3. $d1 = d$ XOR $e1$	// first reduction step
4. $e = d1[74 + (m-1), ..., m]$	// assign part of new vector d to e
5. $e2 = e \times f$	
6. $c = d1$ XOR $e2$	// second reduction step
7. **return** c	

Fig. 3. Classical reduction algorithm

Multiplication in lines 2. and 5. is a short chain of XOR operations. Combining described multiplication and reduction block we have obtained a multiplier with size of 3638 LUTs, frequency of 302 MHz and operation time of 264 clock cycles.

3.2 Montgomery Multiplication Multiplier

The Montgomery algorithm is constructed in a specific way to avoid most costly operations. Instead of performing $c(x) = a(x)b(x) \bmod f(x)$ it performs $c(x) = a(x)b(x)r^{-1}(x) \bmod f(x)$. To perform complete finite-field multiplication $c(x) = a(x)b(x) \bmod f(x)$ one must run the algorithm twice, at first for $a(x)$ and $b(x)$ and then for the obtained result $d(x)$ and $r^2(x) \bmod f(x)$. Operation $c(x) = a(x)b(x) \bmod f(x)$ comprises in fact two steps: 1) $d = MontMult(a, b)$ and 2) $c = MontMult(d, r^2 \bmod f)$ where $MontMult(a, b)$ symbolizes Montgomery multiplication. The algorithm we have used based on Montgomery method [16] calculating the modular product $c(x)$ is as presented in Figure 4.

CONSTANTS: $r(x) = x^m, f(x), f'(x), r^2(x) \bmod f(x)$
INPUT: $a(x), b(x)$
OUTPUT: $c(x) = a(x)b(x) \bmod f(x)$

// $MontMult(a, b)$
1. $t(x) = a(x)b(x)$
2. $u(x) = t(x)f'(x) \bmod r(x)$
3. $d(x) = [t(x)$ `XOR` $u(x)f(x)]/r(x)$
// $MontMult(d, r^2 \bmod f)$
4. $t(x) = d(x)(r^2(x) \bmod f(x))$
5. $u(x) = t(x)f'(x) \bmod r(x)$
6. $c(x) = [t(x)$ `XOR` $u(x)f(x)]/r(x)$
7. **return** c

Fig. 4. Modular multiplication algorithm based on Montgomery method

To be able to utilize the algorithm we need three values, $r(x)$, $r^2(x) \bmod f(x)$, $f'(x)$, which for known irreducible polynomial can be pre-calculated. Element $r(x)$ for field GF(2^m) is chosen to be a simple polynomial x^m (see [12]).

The most complicated in the algorithm is the first step where we need to perform multiplication of two large binary vectors. To perform it we have used multiplier based on shift-and-add method. For $m = 233$ we divide vector b into 16-bit chunks (we add bits equal to 0 on MSB positions of chunk if necessary), multiply sequentially a by all parts of vector b (we need to perform 15 multiplications) and sequentially accumulate partial results. To construct full finite-field multiplier based on Montgomery method, we may use different types of multipliers but we have to remember that they strongly influence final solution.
All other multiplications needed during execution of the algorithm, multiplication by $r(x), r^2(x) \bmod f(x), f'(x), f(x)$ are simpler due to the fact that we know the values of those operands and the number of bits equal to 1 in those polynomials is very low. Thus we may substitute those multiplications with short chains of `XOR` operations. In modulo operation in line 2. and 5., we just cut out all elements of order higher or equal to m. The division operation in line 3. is just simple right shift by m positions. The resulting multiplier uses 2178 LUTs, runs at 323 MHz and needs 270 clock cycles to compute the complete product.

3.3 Mastrovito Multiplication Algorithm

Mastrovito matrix method [15] is an interleaved version of basic matrix-vector approach. In standard matrix-vector approach, we perform two steps: multiplication and reduction with use of special reduction matrix R. Reduction matrix R is a matrix, which coefficients are defined in terms of irreducible polynomial $f(x)$, generating the field. In Mastrovito approach we perform only one step $c = Mb = (A^L + A^H R)b$, where M is so called Mastrovito matrix. Mastrovito matrix M is a combination of A^L, A^H matrices (parts of matrix A representing polynomial a) and special reduction matrix R, see [5].

Matrix M construction and storage is very problematic. Our idea is to partition it into sub-matrices, to save area and ease optimization and synchronization

of operators. The chosen size of sub-matrices is 16×16 bits. In order to save space, coefficients of sub-matrices are not stored in the operator. They are calculated on-the-fly during computations, from parts of matrices A^L, A^H and R (a, b, f operands), by dedicated sub-multipliers.

Sub-multipliers schedule is controlled using a finite-state machine. Results of sub-multiplications are independent of each other and can be calculated in arbitrary order. We can group sub-multipliers in different manners and that way easily change computation time or somehow the design area. We can group sub-matrices into rows, and try to use each sub-multiplier as efficiently as possible; we may change the order of sub-multiplications to adapt the circuit to our needs.

Our basic algorithm for computing the result of $a(x)b(x) \bmod f(x)$ works as follows: the 16×16-bit sub-matrices of M are grouped into rows, spanning one row of matrix M, and for each row 16-bit part of ab product is calculated:

$$\begin{aligned}
c_0 &= M_{(0,0)}b_0 + M_{(0,1)}b_1 + \quad \cdots \quad + M_{(0,m/16)}b_{m/16} \\
c_1 &= M_{(1,0)}b_0 + M_{(1,1)}b_1 + \quad \cdots \quad + M_{(1,m/16)}b_{m/16} \\
&\vdots \\
c_{m/16} &= M_{(m/16,0)}b_0 + M_{(m/16,1)}b_1 + \quad \cdots \quad + M_{(m/16,m/16)}b_{m/16},
\end{aligned}$$

where c_i denotes 16-bit chunk of final result $c(x)$, M denotes Mastrovito matrix resulting from $A^L + A^H R$.

It is easy to observe that there exist many variations of sub-multiplications schedule. The most efficient solution, obtained after many experiments, has the area of 3760 LUTs, frequency of 297 MHz and needs 75 clock cycles to compute the multiplication.

4 Useful Activity Analysis for Multiplication Algorithms

Useful activity of GF(2^m) multipliers has been evaluated using FPGA emulation and activity counters from Section 2. The activity counters are monitoring transitions of each register used in an operator. Corresponding implementation results without and with activity counters are reported in Table 1. The 3× area overhead is due to the numerous counters inserted. Frequency of monitored multipliers is divided by 3. But measurements still can be performed at 100 MHz or more. Such an evaluation speed would not be possible using software simulation.

For all experiments, random operands have been used with uniform and equiprobable distribution for all bits. We performed numerous experiments (corresponding to hundreds of thousands clock cycles for each tested solution). The traces reported below correspond to typical traces. Using average trace is not possible since this may flatten the activity variations and mask information leakage. Thus we have been evaluating our modifications by running modified multipliers for several various sets of experimental data.

Figure 5 (left) presents useful activity measurement results for a typical sequence of GF(2^m) multiplications of random operands using classical algorithm

Table 1. FPGA implementation results of GF(2^m) multipliers without (original operators) and with (monitored operators) activity counters

Algorithms	*without* activity counters			*with* activity counters		
	area LUT	freq. MHz	clock cycles	area LUT	freq. MHz	clock cycles
Classical	3638	302	264	11383	133	264
Montgomery (full)	2178	323	270	6100	121	270
Mastrovito	3760	297	75	5956	110	75

from Section 3.1. There is a high peak at the beginning of each multiplication due to the initialization phase. Figure 5 (right) presents an extract for a single representative multiplication (all random operands lead to the similar overall shape). We have noticed that dependency of the shape of activity variation curves on input data is rather low.

Measurement results for a sequence of random GF(2^m) multiplications using Montgomery algorithm from Section 3.2 are presented in Figure 6 (left) with an extract of a single representative multiplication (right). The reported measurements are shown for complete multiplications with final reduction (for conversion from Montgomery "representation"). We will provide comments on that point at the end of this section. There is a large activity drop at the end of each multiplication due to the reduction step.

Figure 7 (left) presents useful activity measurement results for a typical sequence of random GF(2^m) multiplications using Mastrovito algorithm from Section 3.3 with an extract for a single representative multiplication (right). The variations of the useful activity during a multiplication have a very specific decreasing "step-wave" shape.

Measurements for all three multiplication algorithms show very specific shapes for useful activity variations, which may lead to some information leakage. Those specific shapes provide the attacker with strong temporal references of the operations time location. Based on these references about field-level operations, higher-level operations (e.g. point addition and doubling) can be guessed.

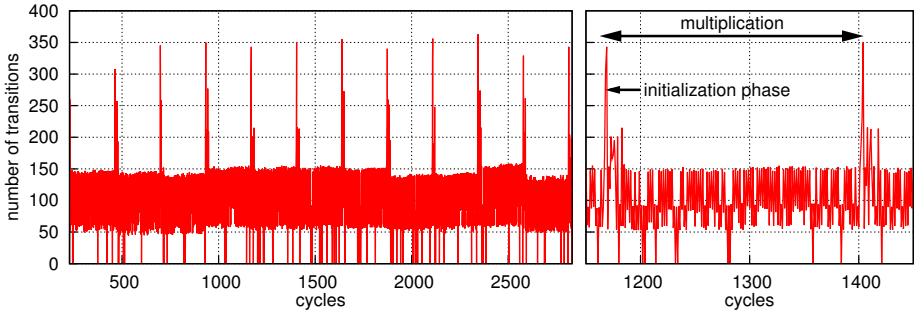

Fig. 5. Useful activity measurement results for random GF(2^m) multiplications with classical algorithm (left). Extract for a single representative multiplication (right).

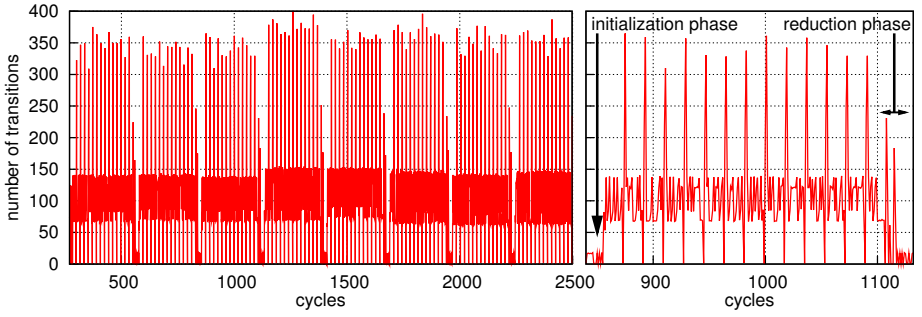

Fig. 6. Useful activity measurement results for random GF(2^m) multiplications with Montgomery algorithm (left). Extract for a single representative multiplication (right).

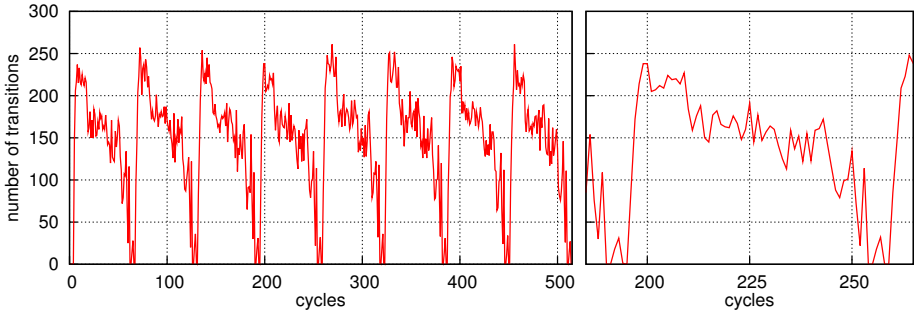

Fig. 7. Useful activity measurement results for random GF(2^m) multiplications with Mastrovito algorithm (left). Extract for a single representative multiplication (right).

Peaks due to the initialization phase at the beginning of operations in Figure 5 are not related to the selected algorithm but to the implemented architecture and especially its control. Resetting all internal registers generates a lot of activity. Then this specific different shape for the initialization phase may occur for other algorithms and architectures. We will see in next section how this problem can be fixed using architecture modifications.

Activity drops at the end of operations in Figure 6 are due to low-complexity reduction step for the considered irreducible polynomial compared to multiplication iterations complexity. We reported measurements for complete multiplication (with final reduction) for fair comparison with other algorithms. In practice, those drops would not appear since reduction is only used at the end of a sequence of operations (with operands in Montgomery domain).

The most problematic shape is the one for Mastrovito algorithm in Figure 7. The decreasing "step-wave" shape is due to variation of the computations quantity in the algorithm. In next section, we will see modifications of this multiplier at algorithmic and arithmetic levels to reduce information leakage.

5 Modifications on Multiplication Algorithms for Reducing Useful Activity Variations

Analyzing the obtained activity variations curves, we can define modification objectives. First, we have to suppress the peaks at the initialization phase. This is an architecture issue (i.e. modification of the operator control). All multiplication algorithms may benefit from this type of modification. Second, we have to take care of the activity drops during the reduction phase of Montgomery algorithm. But as stated in previous section, this phase is only used at the end of long sequence of operations in real ECC applications. Last, we have to make the "step-wave" shape of useful activity variations of Mastrovito algorithm less distinguishable. Below, we describe our modifications for each algorithm.

Classical two-step multiplication: The analysis shows that peaks at the beginning of each multiplication occur due to circuit initialization. To suppress them, we have modified initialization method. Now we do not reset all registers in the first cycle but we have spread the reset activity over several cycles. Figure 8 shows useful activity measurements for a sequence of random multiplications using the modified multiplier. To reduce activity variations, we have also optimized the reduction step by reducing number of registers involved in reduction and merging all the steps of algorithm presented on Figure 3 into a chain of `XOR` operations. In the modified multiplier the average activity varies between 100 and 120 transitions (see Figure 8) while it was about 150 transitions in the original one (see Figure 5). Our modifications reduce the number of active registers in the operator thus they reduce also a little the power consumption of the operator.

Comparing the original operator's useful activity variations (Figure 5) with variations of modified multiplier (Figure 8), we can notice the absence of high initialization peaks. For instance, between cycles 1500 and 2400 it is difficult to detect the executed operations boundaries.

Montgomery multiplication: If we do not consider the reduction step, we may say that the activity variations of Montgomery multiplier are more or less uniform

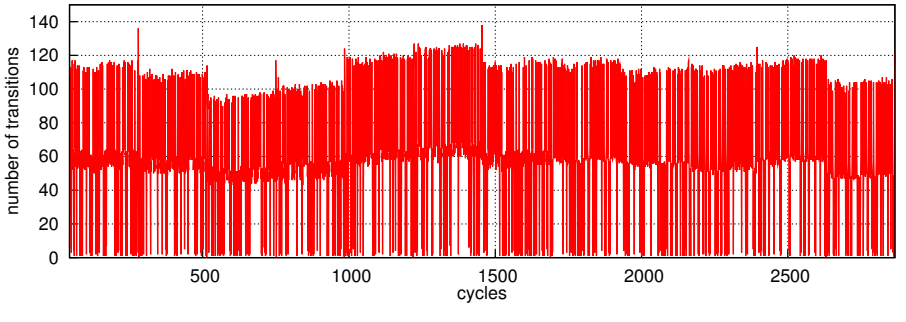

Fig. 8. Useful activity measurement results for random GF(2^m) multiplications with modified classical algorithm

(see Figure 6). The only thing, which may still give some information to the attacker is the initialization phase. Activity drops at this phase occur due to a specific way, in which the input data are fetched. Like for classical algorithm, a modification of the initialization control removes these drops.

Mastrovito multiplication: The "step-wave" shape of useful activity variations of Mastrovito multiplier in Figure 7 is distinguishable and can provide the attacker with a lot of information. Our objective is to modify the algorithm and the architecture in such a way that multiplications cannot be too easily distinguished.

We have investigated two types of modifications for Mastrovito multiplier: "uniformization" of the number of sub-multipliers' registers used in each clock cycle and "randomization" of the starting times of the operator sub-multipliers. We have derived many versions of those two types of modifications. The most worth showing according to us are variations of "randomization".

Figure 9 presents the way we have divided matrix M into sub-matrices (see Section 3.3). The boxes with same indices Mi denote blocks, which can be multiplied by parts of b, using the same sub-multiplier module.

M	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M1	M1	M1	M1	Mc
1	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M1	M1	M1	Mc
2	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M1	M1	Mc
3	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M1	Mc
4	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	M4	M4	M4	M4	Mc
5	M2	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	M4	M4	M4	Mc
6	M0	M2	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	M4	M4	Mc
7	M0	M0	M2	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	M4	Mc
8	M0	M0	M0	M2	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	Mc
9	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	M3	M3	M3	M3	Mc
10	M0	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	M3	M3	M3	Mc
11	M0	M0	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	M3	M3	Mc
12	M0	M0	M0	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	M3	Mc
13	M0	M0	M0	M0	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	Mc
14	Mr	Mr	Mr	Mr	Mr	Mr	Mr	Mr	Mr	Mr	Mr	Mr	Mr	Mr	Mr

Fig. 9. Mastrovito matrix for $m = 233$

It can be observed that some sub-multipliers are used more than the others. Thus if we start them all at the same time, the activity is higher at the beginning of the operation (where all sub-multipliers are used) and smaller at the end (almost all sub-multipliers are already switched off). Thus our first proposition is to make the utilization, in one clock cycle, of the number of internal registers

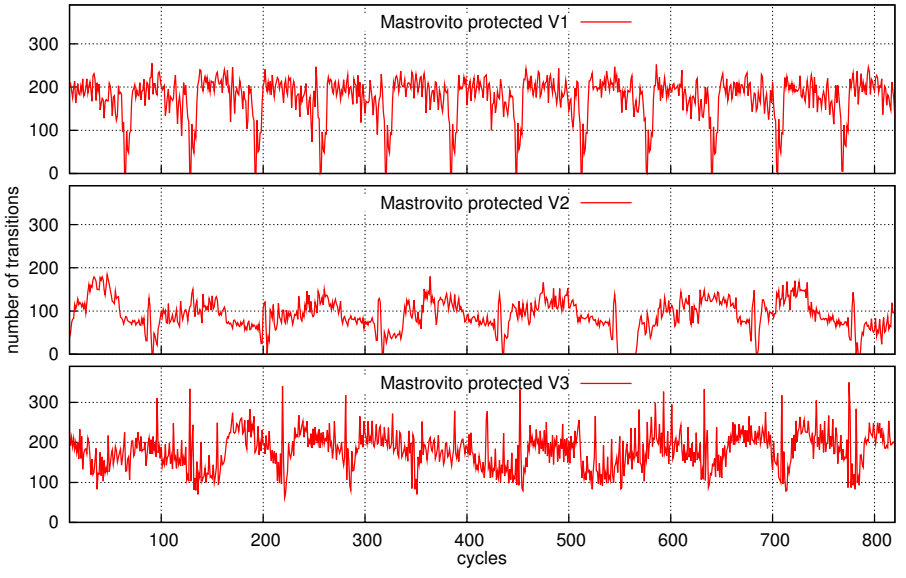

Fig. 10. Useful activity measurement results for random GF(2^m) multiplications with 3 versions of modified Mastrovito algorithm.

more uniform. For this we have modified the schedule of the sub-multipliers work but without changing the complete computation time. We have tried several schemes and the best yet obtained leads to the useful activity variations curve presented in Figure 10 (version v1 on top curve).

Activity shape for this modified Mastrovito multiplier shows small improvement. However we still try to improve our way of uniformization of utilization of internal registers used, taking into account also their sizes (each sub-multiplier uses different number of registers of various sizes). The dummy operations (shifting, incrementation) performed on some of those registers does not give visible improvements of activity variations shape. The dependencies between the sub-computations, limit the efficiency of this modification.

Our next objective was to randomize the starting moment of each sub-multiplier. This should "spread more" the activity over the whole computation. In order to randomize the beginning of sub-multiplications, we have used 8-bit LFSR register (Mastrovito v2) and pseudo random generator based on 4-bit LFSR (Mastrovito v3), which initialization values depend on some bits of a and b operands. In order to avoid blocking the multiplier we exchange the initialization values (seed) many times throughout multiplication operation. We have also tried other methods but the best results so far were achieved with use of LFSRs. Due to the randomization, the time needed to perform the complete multiplication, depending on which sub-multiplier is started first, will either decrease or increase randomly. The average number of clock cycles for Mastrovito v2 is 116 (minimal value: 98, maximal value: 126), whereas for Mastrovito v3

Table 2. FPGA implementation results of modified GF(2^m) multipliers with reduced activity variations

Algorithms	balanced		area		speed		#
	area LUT	freq. MHz	area LUT	freq. MHz	area LUT	freq. MHz	clock cycles
Classical	2868	270	2778	228	3444	420	260
$\times\alpha$ factor	$\times$0.79	$\times$0.89	$\times$0.76	$\times$0.75	$\times$0.95	$\times$1.39	$\times$0.98
Montgomery	2099	323	2093	338	2099	423	264
$\times\alpha$ factor	$\times$0.96	$\times$1.00	$\times$0.96	$\times$1.05	$\times$0.96	$\times$1.31	$\times$0.98
Mastrovito v1	3463	414	3439	343	3489	384	75
$\times\alpha$ factor	$\times$0.99	$\times$1.50	$\times$0.98	$\times$1.24	$\times$0.94	$\times$1.39	$\times$1.00
Mastrovito v2	3700	306	3667	253	3717	388	avg. 116
$\times\alpha$ factor	$\times$1.06	$\times$1.11	$\times$1.05	$\times$0.92	$\times$1.06	$\times$1.41	$\times$1.55
							min.98, max.126
Mastrovito v3	3903	319	3837	250	4335	375	avg. 80
$\times\alpha$ factor	$\times$1.12	$\times$1.16	$\times$1.10	$\times$0.91	$\times$1.24	$\times$1.36	$\times$1.07
							min.64, max.108

average number of clock cycles needed is 80 (minimal value: 64, maximal value: 108). Useful activity measurements for v2 (middle curve) and v3 (bottom curve) modifications are presented in Figure 10. As one can observe on Figure 10, the shapes of useful activity variations are more irregular and not easily predictable compared to the curve for the initial version in Figure 7.

Implementation results for the modified multipliers: All modified multiplication algorithms have been implemented in FPGA. The corresponding results are reported in Table 2. Three optimization targets were used for the synthesis tool: balanced area/speed, area and speed optimizations. In order to compare the modified multipliers to the original ones (data from Table 1), we report a comparison factor α such as $\texttt{modified} = \alpha \times \texttt{original}$ both for area and frequency.

Evaluation of activity variation reduction: We used signal processing tools to evaluate our modifications. The measured activity traces, in time domain, are transformed into frequency domain using FFT (Fast Fourier Transform), see [20]. Figure 11 presents those results for unprotected and protected versions of some multipliers. It represents the mathematical power for each frequency bin and Y-axis uses the same logarithmic scale for all versions. Figure 11 shows an important reduction in the potential information leakage for all frequencies.

In order to numerically compare solutions, we have computed the spectral flatness measure (SFM) [20]:

$$\mathrm{SFM} = \frac{\sqrt[n]{\prod_{i=1}^{n} p(i)}}{\frac{1}{n}\sum_{i=1}^{n} p(i)} \qquad \in [0,1]$$

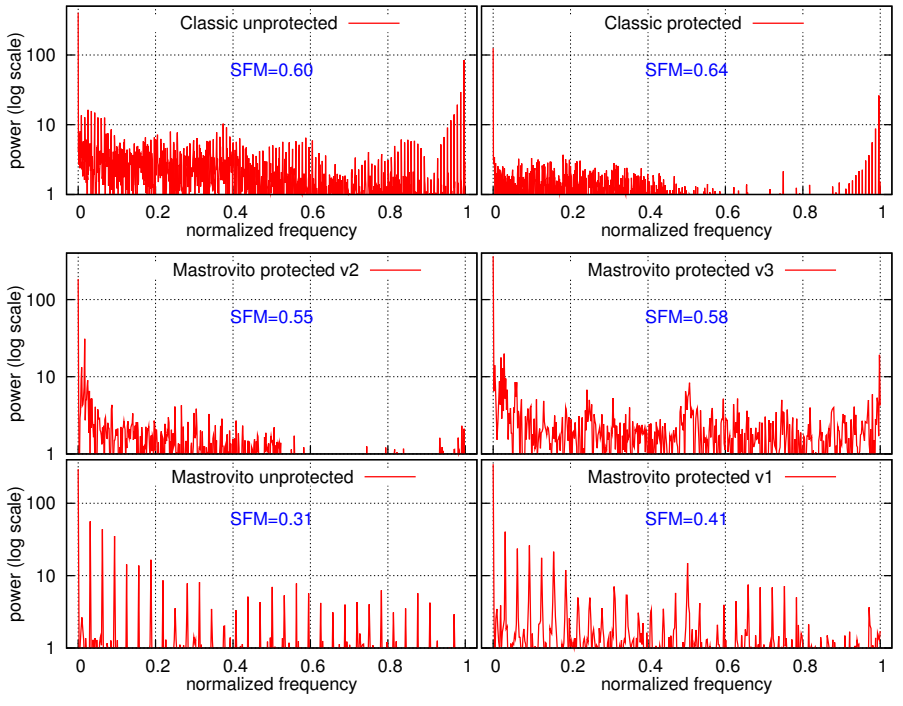

Fig. 11. FFT analysis results for unprotected and protected versions of multipliers (top: classic algorithm, middle and bottom: Mastrovito algorithm for various versions)

SFM is the ratio of the geometric mean to the arithmetic mean for a collection of n frequency bins $p(i)$ (power for frequency bin i). A SFM close to 1 indicates a spectrum with power well distributed in all frequency bins (flat curve) while a SFM close to 0 indicates that power is concentrated into a few bins (curve with peaks). SFM values are reported on Figure 11. Improvement is limited for classic algorithm, but for Mastrovito, our modifications lead to significant improvement (from 0.31 for unprotected version to 0.58 for the best protected version).

The obtained results are rather satisfying. We can see that there is a way to reduce information leakage. In the future, we plan to study chain of operations in ECC primitives such as point addition, point doubling and scalar multiplication.

6 Conclusion

GF(2^m) multipliers with reduced useful activity variations have been proposed. Useful activity has been evaluated using accurate FPGA emulation and activity counters at every operation cycle. Measurement analysis shows that the implemented multiplication algorithms (classical, Montgomery and Mastrovito) lead to specific shapes for the curve of activity variations which may be used as a small source of information leakage for some side channel attacks.

We proposed modifications of selected GF(2^m) multipliers to reduce this information leakage source at two levels: architecture level by removing activity peaks due to control (e.g. reset at initialization) and algorithmic level by modifying the shape of the activity variations curve. Due to very low-level optimizations there is no significant area and delay overhead.

We have to complete our theoretical analysis with physical measurements and parasitic activity effects to get accurate results. We will study similar issues for very advanced GF(2^m) multiplication algorithms such as [23,6,17,1] and for other operations (e.g. addition, subtraction, inversion, multiplication by constant and scalar multiplication) used in ECC protocols.

Acknowledgment. This work has been supported in part by a French government scholarship (in French: *bourse du gouvernement français*).

References

1. Bajard, J.-C., Negre, C., Plantard, T.: Subquadratic space complexity binary field multiplier using double polynomial representation. IEEE Transactions on Computers 59(12), 1585–1597 (2010)
2. Byrne, A., Meloni, N., Tisserand, A., Popovici, E.M., Marnane, W.P.: Comparison of simple power analysis attack resistant algorithms for an elliptic curve cryptosystem. Journal of Computers 2(10), 52–62 (2007)
3. Chabrier, T., Pamula, D., Tisserand, A.: Hardware implementation of DBNS recoding for ECC processor. In: Proc. 44th Asilomar Conference on Signals, Systems and Computers, pp. 1129–1133. IEEE (November 2010)
4. Deschamps, J.-P., Imana, J.L., Sutter, G.D.: Hardware Implementation of Finite-Field Arithmetic. McGraw-Hill (2009)
5. Erdem, S.S., Yanik, T., Koc, C.K.: Polynomial basis multiplication over GF(2^m). Acta Applicandae Mathematicae 93(1-3), 33–55 (2006)
6. Fan, H., Hasan, M.A.: A new approach to subquadratic space complexity parallel multipliers for extended binary fields. IEEE Transactions on Computers 56(2), 224–233 (2007)
7. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
8. Guo, X., Schaumont, P.: Optimized system-on-chip integration of a programmable ECC coprocessor. ACM Transactions on Reconfigurable Technology and Systems 4(1), 6:1–6:21 (2010)
9. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer (2004)
10. Joye, M.: Defenses Against Side-Channel Analysis. In: Advances in Elliptic Curve Cryptography. London Mathematical Society Lecture Note Series, vol. 317, pp. 87–100. Cambridge University Press (April 2005)
11. Karatsuba, A., Ofman, Y.: Multiplication of multi-digit numbers on automata. Doklady Akad. Nauk SSSR 145(2), 293–294 (1962) (in Russian); Translation in Soviet Physics-Doklady 44(7), 595–596 (1963)
12. Koc, C.K., Acar, T.: Montgomery multiplication in GF(2^k). Designs, Codes and Cryptography 14(1), 57–69 (1998)

13. Lidl, R., Niederreiter, H.: Introduction to Finite Fields and Their Applications, 2nd edn. Cambridge University Press (1994)
14. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer (2007)
15. Mastrovito, E.: VLSI Architectures for Computation in Galois Fields. PhD thesis, Department of Electrical Engineering, Linkoping University, Sweden (1991)
16. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
17. Namin, A.H., Huapeng, W., Ahmadi, M.: A high-speed word level finite field multiplier in F_{2^m} using redundant representation. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 17(10), 1546–1550 (2009)
18. Orlando, G., Paar, C.: A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m). In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 41–56. Springer, Heidelberg (2000)
19. Oswald, E.: Side Channel Analysis. In: Advances in Elliptic Curve Cryptography. London Mathematical Society Lecture Note Series, vol. 317, pp. 69–86. Cambridge University Press (April 2005)
20. Proakis, J.G., Manolakis, D.G.: Digital Signal Processing. Prentice Hall (1996)
21. Rodriguez-Henriquez, F., Saqib, N.A., Diaz-Perez, A., Koc, C.K.: Cryptographic Algorithms on Reconfigurable Hardware. Springer (2007)
22. Savas, E., Koc, C.K.: Finite field arithmetic for cryptography. IEEE Circuits and Systems Magazine 10(2), 40–56 (2010)
23. Sunar, B.: A generalized method for constructing subquadratic complexity GF(2^k) multipliers. IEEE Transactions on Computers 53(9), 1097–1105 (2004)
24. Tisserand, A.: Low-power arithmetic operators. In: Piguet, C. (ed.) Low Power Electronics Design, ch. 9. CRC Press (November 2004)
25. Tisserand, A.: Fast and accurate activity evaluation in multipliers. In: Proc. 42nd Asilomar Conference on Signals, Systems and Computers, pp. 757–761. IEEE (October 2008)
26. Weste, N.H.E., Harris, D.: CMOS VLSI Design: A Circuits and Systems Perspective, 3rd edn. Addison Wesley (2004)