Full Hardware Implementation of Short Addition Chains Recoding for ECC Scalar Multiplication

Julien Proy, Nicolas Veyrat-Charvillon, Arnaud Tisserand, Nicolas Méloni

Compas'2015 : Parallélisme/ Architecture / Système
Lille, France, du 30 juin au 3 juillet 2015

Full Hardware Implementation of Short Addition Chains Recoding for ECC Scalar Multiplication

Julien Proy[1], Nicolas Veyrat-Charvillon[3,1], Arnaud Tisserand[2,1], and Nicolas Méloni[4]

[1]IRISA, [2]CNRS – [3]Université Rennes 1 – INRIA. 6 rue de Kerampont, 22305 Lannion.
[4]University Toulon, France

Résumé
Ensuring uniform computation profiles is an efficient protection against some side channel attacks (SCA) in embedded systems. Typical elliptic curve cryptography (ECC) scalar multiplication methods use two point operations (addition and doubling) scheduled according to secret scalar digits. Euclidean addition chains (EAC) offer a natural SCA protection since only one point operation is used. Computing short EACs is considered as a very costly operation and no hardware implementation has been reported yet. We designed an hardware recoding unit for short EACs which works concurrently to scalar multiplication. It has been integrated in an in-house ECC processor on various FPGAs. The implementation results show similar computation times compared to non-protected solutions, and faster ones compared to typical protected solutions (e. g. 18 % speed-up over 192 b Montgomery ladder).

Mots-clés : elliptic curve cryptography ; scalar multiplication ; side channel attack ; countermeasure ; addition chains ;

1. Introduction

Elliptic curve cryptography (ECC [5]) has become the recommended standard in asymmetric protocols, as it provides equivalent security levels as RSA with smaller keys and benefits from more efficient implementations. The main operation in ECC protocols is the *scalar multiplication* $[k]P$, where a specific curve point P is multiplied by a large integer k (secret/public key). Most scalar multiplication algorithms perform a sequence of curve-level operations that depends on the k_i (the bits or recoded digits of k). For instance, the basic double-and-add method either performs a point doubling (`DBL`) or a `DBL` followed by a point addition (`ADD`) depending on each k_i. Such behavior is an issue when considering *side channel attacks* (SCAs [11]). Basically, an attacker can perform physical observations of computations (e.g. the power consumption of the device) and exploit them to recover the secret key being used in the device. ECC operations should be protected against SCAs in many applications, especially in embedded systems. Several attacks [14] and countermeasures [8] have been proposed for ECC scalar multiplication. Countermeasures typically work by hiding or confusing the relation between the physical behavior of a device and the secret key being used.
One proposal for a countermeasure is to perform scalar multiplication by using *addition chains* [9] where a single curve point operation is used (i.e. `ADD`), providing intrinsic resilience to *simple power analysis* (SPA). The scalar k has to be *recoded* into an addition chain $\mathcal{C}$. At each step of the

scalar multiplication, the operands of a new ADD are some of the previously computed results or P. For speed purposes, the more specific *euclidean addition chains* (EAC) are used [12], allowing for a faster ADD. Then the number of ADDs is directly related to $\mathcal{C}$ length. Up to now, it was generally admitted that computing a *short* EAC is too expensive to make EAC competitive with other SPA secure methods[12, 6]. So far, it appears that the complexity of efficient recoding has precluded any attempt at an hardware implementation of EAC multiplication.
In this work, we propose a complete *hardware implementation* of a *recoding unit* for short EACs and include it into an FPGA implementation of an ECC processor. Our unit allows efficient EACs to be computed without incurring an additional cost for the scalar multiplication since it works *concurrently* to the scalar multiplication operations. Section 2 quickly recalls ECC background and details EACs recoding methods. Details and implementation results about our recoding unit are reported in Section 3. In Section 4, we compare our unit to other common recoding methods (all integrated using the same ECC processor framework) and to other rare works with practical results. Finally, Section 5 concludes the paper.

2. State-of-the-Art

2.1. Definitions

Definition 1 *Let* k *be an integer. An addition chain for* k *is a sequence of integers* $a_0, a_1, \ldots, a_l$ *satisfying :* $a_0 = 1$, $a_l = k$, *and* $a_i = a_{i_1} + a_{i_2}$ *for some* $i_2 \leq i_1 < i$.

Definition 2 *A euclidean addition chain (EAC) for* k *satisfies the additional condition : for* $i \geq 3$, $a_i = a_{i_1} + a_{i_2} \Rightarrow (a_{i+1} = a_i + a_{i_1}$ *or* $a_{i+1} = a_i + a_{i_2})$

Indeed each a_{i+1} is obtained by adding a_i to one of the terms of $a_i = a_{i_1} + a_{i_2}$. This reduces the access to previous values. If the largest summand is added, this is referred to as a *big step* and noted 0 (respectively, adding the smallest summand is taking a *small step* noted 1). This way, an EAC can be translated into a binary number.
One interesting property of this binary representation of EAC is that the reversed chain computes the same integer [6] as illustrated in Figure 1 for $k = 14$.

2.2. Securing Scalar Multiplication Against SPA

EACs provide a natural way of protecting scalar multiplication against some side channel attacks, in particular *simple power analysis* (SPA, see [11]), since a single curve operation (point addition) is used. The scalar multiplication method based on EAC is presented in Figure 2. It works by accumulating multiples of P, starting from P and [2]P, in an order determined by the EAC such that the end result is [k]P. The same curve operation (namely an ADD) is performed with every iteration, and the only dependence to the key bits is on which operand (either U_1 or U_2) is kept in memory. This makes the algorithm resilient to simple power analysis.
The Montgomery ladder (ML [13]) and unified formulas (UF [2]) are other SPA resilient algorithms that rely on using a single curve operation. They all ensure that the power profile of

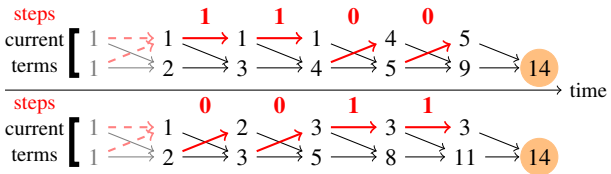

FIGURE 1 – Examples of EACs for computing $k = 14$.

EAC Point Multiplication : $\mathcal{C} = \mathrm{EAC}(k)$, point P

1: $(U_1, U_2) \leftarrow (P, [2]P)$
2: **for** $c_i \in \mathcal{C}$ **do**
3: **if** $c_i = 0$ **then** $(U_1, U_2) \leftarrow (U_2, U_1 + U_2)$ (*big step*)
4: **else** $(U_1, U_2) \leftarrow (U_1, U_1 + U_2)$ (*small step*)
5: **return** $Q = U_1 + U_2$ $(Q = [k]P)$

FIGURE 2 – EAC-based scalar multiplication algorithm

a scalar multiplication (which field operations are performed and when) does not depend on key digits. ML always performs both point operations (ADD then DBL), whereas UF consist in performing a sequence of field operations that works for both ADD and DBL.
Table 1 reports the cost of different methods for scalar multiplication in terms of field multiplications per key bit or chain bit. It appears that the efficiency of protected algorithms tends to be less than that of non-protected methods. In order to be comparable with unprotected methods, the EAC recoding of an n-bit scalar k should have a length l shorter than $2n$.
An EAC for a given integer k is obtained by applying the subtractive version of Euclid's algorithm. The second input to Euclid's algorithm, an integer co-prime to k noted g in the following, strongly impacts the length of the EAC. A random choice tends to lead to very long chains (average length $O(\ln(k)^2)$[13]), which means a slow scalar multiplication.

2.3. Efficient EAC recoding

Finding a short EAC for a given scalar k is a random search process, where numerous starting values for the first residue g are tested. A good heuristic is to look around $g = k/\phi$, where ϕ is the golden ratio [13]. This choice ensures that the end of the EAC is comprised of big steps, more efficient by nature. Accurate bounds on their number are given in [7, corollary 1]. For instance, given a 192-bit scalar, recoding values with $g \in \left[\frac{k}{\phi} - 500, \frac{k}{\phi} + 500\right]$ (meaning a range search of 1000) guarantees that the EAC ends with 131 big steps. Combined with the reversal property of EAC recoding, this allows us to compute the first (big) steps of the scalar multiplication while performing several tentative EAC recodings. Our approach consists in the following steps :

1. for a given k, compute $\frac{k}{\phi}$ (multiplication by the constant $\frac{1}{\phi}$),
2. choose a range width 2ε and compute the minimal number of starting big steps m,
3. in parallel :
   – start the EAC scalar multiplication with m big steps,
   – try all values of g in the interval $[\frac{k}{\phi} - \varepsilon, \frac{k}{\phi} + \varepsilon]$ to find a short EAC recoding of k,
4. finish the scalar multiplication using the shortest EAC.

We have performed simulations to measure the ratio between the EAC length l and the scalar size n in function of the range width 2ε (i.e. nb. values tested around k/ϕ). Figure 3 reports our

TABLE 1 – Cost of scalar multiplication for several methods

Method	DA	NAF-3	NAF-4	ML	UF	EAC
Source	[5]	[5]	[5]	[13]	[2]	[12]
Cost (M/bit)	17	13.5	12.8	24	19	7*

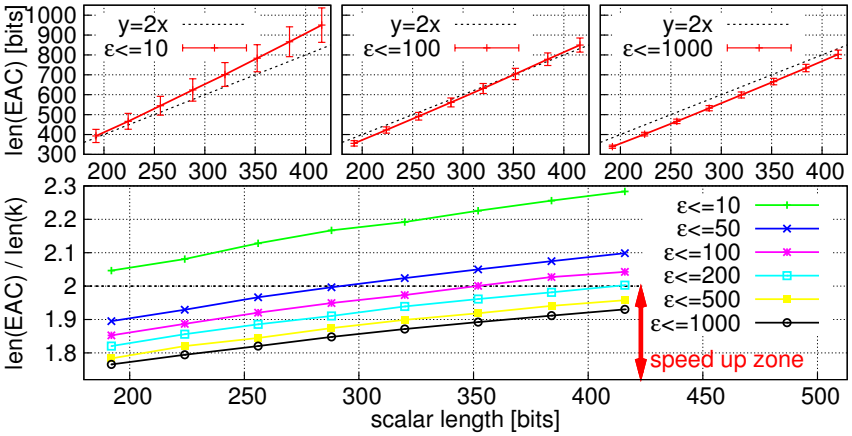

FIGURE 3 – Statistical analysis of short EAC recoding.

results for the lengths of the shortest EAC found by exploring $[\frac{k}{\phi}-\varepsilon, \frac{k}{\phi}+\varepsilon]$, averaged over 1000 random scalars.

3. Proposed Short EAC Recoding Unit

We designed a complete and standalone hardware recoding unit for short EACs based on the method presented above. To the best of our knowledge, it appears to be the first hardware implementation of EAC recoding. Below, we describe our unit and the selection of its internal parameters. Then, we report detailed implementation results on various FPGAs and for several key sizes (n). For comparison purposes, we also report implementation results for other existing recoding methods on the same FPGAs. We were only able to identify one reference with complete hardware implementation results (but for another type of recoding) used in ECC scalar multiplication.

3.1. EAC Recoding Unit Architecture

The architecture of our recoding unit is described in Figure 4. For scalability, small delay and simplicity purposes, we split all large (n-bit) values into small w-bit words (i.e. $w \ll n$). The unit is centered around a single memory block (BRAM at top-left) which stores all the elements used in the recoding : the key k, the inverse golden ratio $1/\phi$ (constant), the scaled key k/ϕ, the two remainders used in the subtractive euclidean algorithm a and b, the current addition chain $\mathcal{C}$ and the shortest chain found so far, $\mathcal{C}'$. The address of each element in the memory is the sum of the word index and the element offset (e.g. 0 for $1/\phi$, offset k, offset $k/\phi\ldots$).

Key recoding works the following way : input k is first multiplied by $1/\phi$ using an adder (parallel-serial multiplication at top-right), and the result k/ϕ is memorized. The unit then iterates over values of ε (stored in a small register), performing a subtractive euclidean algorithm on values $a = k$ and $b = k/\phi \pm \varepsilon$. The successive remainders a and b are subtracted from one another using two adders ($a^{(j)}-b^{(j)}$ and $b^{(j)}-a^{(j)}$ at bottom left), and a comparison of the new remainders is performed simultaneously ($\frac{a^{(j)}}{2}-b^{(j)}$ and $\frac{b^{(j)}}{2}-a^{(j)}$ at bottom right). Only the carry out of these two last subtracters is used to select the new recoded term (small step or big step) in $\mathcal{C}$. The recoded bits of successive comparisons are concatenated locally in a w-bit serial-in parallel-out (SIPO) register to provide chain $\mathcal{C}$Ònce the algorithm terminates (i.e. either remainder reaches zero) the chain $\mathcal{C}$ is memorized, provided it is shorter than $\mathcal{C}'$ the best chain so far (a small register in the control stores the smallest chain length).

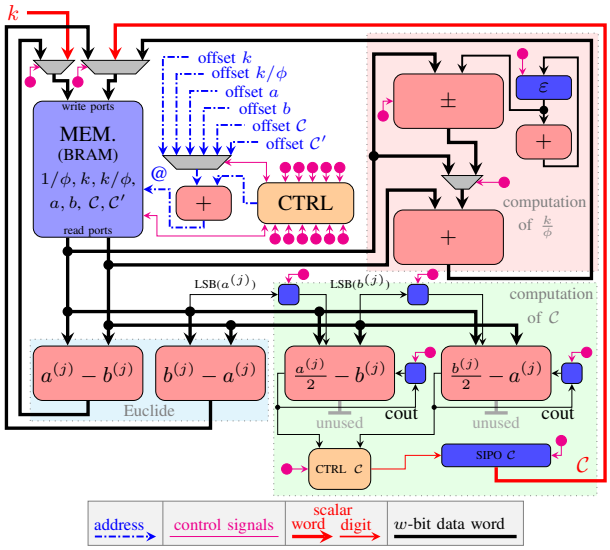

FIGURE 4 – Architecture of the proposed recoding unit.

3.2. Parameters Selection

In order to keep an efficient scalar multiplication, we target addition chains with length at most $2n$ (e.g. chains of length 384 bits/steps for a 192-bit key). Based on the results presented in Figure 3, this seems to be a reasonable constraint.

Our unit has been integrated for tests into an existing ECC processor (targeted for small area on low-cost FPGAs), with internal word size 32-bit. To simply the external interface, we used $w = 32$ bits within the recoding unit.

Our recoding unit requires just one single BRAM (18 Kb or 36 Kb for Xilinx FPGAs) for all sensible key sizes (e.g. 160–600 bits). For a large key size n_{max}, the memory has to store at most $9 \times n_{max}$ bits ($1 \times n_{max}$ for $1/\phi$, k, k/ϕ, a, b and $2 \times n_{max}$ for C and C'). In practice $n_{max} < 600$, so the maximum memory size is at most 5.4 Kb.

Our goal is to show the feasibility of EAC recoding in hardware. We tried to assume as little as possible about the available hardware cells, and did not use hardwired DSP blocks present in Xilinx FPGAs. Those DSP blocks are only useful during the key scaling step (multiplication by $1/\phi$) since we are using a subtractive euclidean algorithm (which only requires additions/subtractions).

3.3. Implementation Results on Various FPGAs

We implemented our recoding unit for several key sizes commonly used in embedded systems ($n \in \{160, 192, 256, 384\}$ bits) and on various FPGAs (both low-cost/high-performances and recent/older devices). We also tried to evaluate the impact of optimization (area or speed) targets in the ISE 14.1 Xilinx environment. Table 2 reports detailed implementation results for recoding units on a Spartan-6 LX9 FPGA and the 4 key sizes. Reporting durations for a standalone unit is not relevant since its real speed will be impacted by the crypto-processor. Timing performances for a complete scalar multiplication are reported in Section 4.

In [3], Virtex-5 LX50T implementation results are reported for a recoding unit based on multiple base number system (MBNS). We also provide implementation results for our unit on this specific FPGA in the bottom of Table 2. We also report a few implementation results for other FPGAs in Table 3 (due to page limit, we only report $n = 192$ bits).

TABLE 2 – FPGA implementation results of dedicated units for various recoding methods (on XC6SLX9/XC5VLX50T).

FPGA	Rec. Meth.	n	BRAM	optimization : area		optimization : speed	
				area slices (FF/LUT)	freq. MHz	area slices (FF/LUT)	freq. MHz
Spartan6	EAC	160	1	**209** (636/441)	**151**	211 (662/476)	151
		192	1	**195** (641/441)	**160**	223 (634/476)	157
		256	1	**203** (636/441)	**155**	214 (662/476)	154
		384	1	228 (652/441)	154	**215** (682/476)	**159**
Spartan6	Bin. & ML	160	1	**26** (70/101)	**466**	73 (194/237)	388
		192	1	**26** (70/101)	**466**	75 (231/270)	387
		256	1	**26** (70/101)	**466**	94 (300/336)	377
		384	1	**26** (70/101)	**466**	128 (446/475)	379
Spartan6	NAF-3	160	1	35 (104/122)	328	**34** (108/130)	**382**
		192	1	35 (104/122)	328	**34** (108/130)	**382**
		256	1	35 (104/122)	322	**34** (108/130)	**364**
		384	1	42 (120/123)	248	**43** (123/131)	**332**
Spartan6	NAF-4	160	1	40 (109/125)	333	**36** (113/134)	**388**
		192	1	40 (109/125)	333	**36** (113/134)	**388**
		256	1	40 (109/125)	333	**36** (113/134)	**365**
		384	1	45 (129/126)	236	**43** (132/135)	**320**
Virtex5	EAC	160	1	278 (711/509)	198	**276** (709/476)	**206**
	NAF-4	160	1	**60** (148/161)	**418**	61 (147/157)	413
	MBNS [3]	160	1	153 (301/412)	232	323 (682/908)	202
				$n/12 + 4$ clock cycles		$n/24 + 4$ clock cycles	

4. Comparison to Other Methods

4.1. Methodology

Comparing implementation results of recoding units without integrating them in a real ECC processor is not sufficient. Our short EAC recoding unit and other units for commonly used recoding methods have been integrated in an in-house compact ECC processor (designed for low-cost embedded applications and will be distributed as open source in the future). The global architecture of this processor is shown in Figure 5. The processor operates using several dedicated arithmetic units (field adder, multiplier and inverter). The control flow in the processor is microcoded for flexibility reasons. The microcode is stored in a dedicated BRAM ("Prg. Mem." on left). Another BRAM stores the coordinates of curve points handled during computations ("Points Mem." on right). The selected recoding unit (top left) outputs recoded key digits (k_i) that are used during microcode execution to alter the program flow.

4.2. Description of the Compared Methods

We compared our short EAC recoding unit with several representative recoding methods for ECC scalar multiplication :

– The basic *double-and-add* (DA) (k_i digits are directly used without any recoding [5, Sec. 3.3.1]) which provides a reference point but without any SPA protection.
– The more efficient *non-adjacent form* with window (NAF-w, a specific kind of signed-digit recoding [5, Sec. 3.3.1]) methods with NAF-3 and NAF-4 versions, which are faster but do not offer any protection against SPA.

TABLE 3 – Implementation results of recoding units on various FPGA for $n = 192$ bits (XC3S50, XC4VLX40, XC7A100T, XC7VX330T).

FPGA	Recoding Method	optimization : area		optimization : speed	
		area slices (FF/LUT)	freq. MHz	area slices (FF/LUT)	freq. MHz
Spartan 3	EAC	618 (911/541)	79	634 (911/574)	81
	Bin. & ML	73 (108/133)	265	75 (110/134)	240
	NAF-3	124 (193/154)	151	131 (197/161)	185
	NAF-4	133 (210/157)	154	140 (214/165)	150
Virtex 4	EAC	565 (868/477)	170	597 (899/511)	191
	Bin. & ML	73 (108/133)	404	162 (285/268)	409
	NAF-3	120 (189/154)	264	127 (192/161)	379
	NAF-4	128 (205/157)	266	135 (210/165)	375
Artix 7	EAC	227 (637/435)	151	277 (667/471)	145
	Bin. & ML	25 (86/101)	453	27 (86/102)	478
	NAF-3	37 (120/122)	319	43 (124/125)	418
	NAF-4	51 (125/125)	311	42 (130/129)	415
Virtex 7	EAC	226 (642/434)	215	281 (762/468)	216
	Bin. & ML	25 (86/101)	435	67 (220/270)	412
	NAF-3	40 (119/122)	413	47 (123/125)	415
	NAF-4	48 (124/125)	417	46 (128/129)	472

– The Montgomery ladder (ML [13]) enforces uniform sequences of operations against SPA.
– The unified formulas (UF) from [2].
– Our short EAC recoding unit presented in this work.

They have been implemented in dedicated units and integrated in our ECC processor. DA, ML and UF methods share identical hardware configurations (only the microcode differs).

4.3. Comparison Results and Analysis

The ECC processor has been synthesized, placed and routed with each recoding unit. Table 4 reports area values with number of slices, flip-flops (FFs) and look-up tables (LUTs), as well as operating frequencies and average durations for a complete scalar multiplication. Due to page limit, Table 4 only presents results for $n = 192$ bits and Spartan-6 FPGA (similar behavior is obtained for other n and FPGAs). Our solution with short EAC recoding is 25 % faster than the commonly used Montgomery ladder for an area overhead of 39 %. This area overhead is important, but one should keep in mind that our ECC processor is a very small one. Integrating an advanced recoding unit impacts a lot the total area. We expect to get a smaller overhead for larger processors.

Figure 6 illustrates the impact of the EAC recoding process which works concurrently to the beginning of the scalar multiplication. As described in Section 2, the first bits of the reversed chain always correspond to big steps (there is a theoretical minimal number of big steps [7]) and determines a maximum recoding effort. The recoding must be finished in the worst case before the theoretical deadline (about 40 % of the complete scalar multiplication for our parameters). We recoded more than 10000 scalars for each key size n. Results presented left column are normalized to the duration of the ML version (for similar SPA protection). The speed-up is in 20–25 %. The right column presents the probability that a recoding process is not finished when the deadline arrives. For small n, our EAC recoding unit is fast enough the ensure about 25 % average speedup compared to ML. But for $n = 384$ is 11 % of cases the recoded chain is slightly

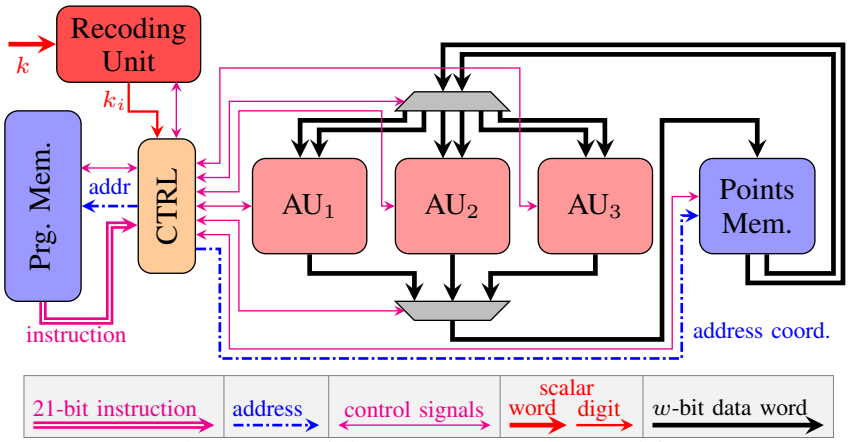

FIGURE 5 – Architecture of the ECC processor used for comparisons.

TABLE 4 – FPGA implementation results of the ECC processor with various recoding methods for $n = 192$ bits (on Spartan-6 XC6SLX9).

recoding method	BRAM	optim. target	area slices (FF/LUT)	freq. MHz	dura. ms	SCA prot.
EAC	3	area	534 (1813/1508)	132	35.8	Y
		speed	556 (1872/1523)	137	34.5	
DA	2	area	429 (1243/1134)	191	30	N
		speed	399 (1302/1222)	177	32.5	
ML	2	area	429 (1243/1134)	191	42.5	Y
		speed	399 (1302/1222)	177	45.8	
UF	2	area	429 (1243/1134)	191	50.4	Y
		speed	399 (1302/1222)	177	54.4	
NAF-3	2	area	422 (1280/1157)	181	25.2	N
		speed	423 (1321/1242)	175	26.1	
NAF-4	2	area	420 (1277/1161)	158	27.3	N
		speed	425 (1233/1246)	177	24.4	

larger than $2n$. This reduces the average speedup to 20 %.
Few previous works present complete hardware implementation results for advanced recoding schemes against SCAs for ECC scalar multiplication, see Table 5. Comparing all those solutions with ours is quite complex since several works used a large number of hardwired DSP blocks.

5. Conclusion

In this work, we report the first hardware implementation of euclidean addition chains for ECC. Hardware implementation of EAC recoding is efficient thanks to some mathematical properties : the insurance of ending the chain by several big steps, the reversibility of chains and the efficiency of addition formula. Our prototype implementation adds a limited area overhead and provides speed improvements over protected schemes such as Montgomery Ladder and unified formulas.
We identified possible optimizations for future work. We are faithful that this unit will be competitive in terms of area and speed even with non-protected methods such as double-and-add

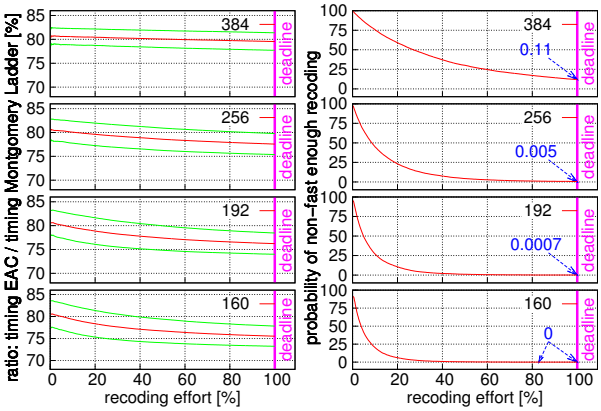

FIGURE 6 – Statistical behavior of the proposed EAC recoding unit (normalized to ML). Left column shows average (middle curve) plus/minus standard deviation (top/bottom curves). Right column shows the probability of recoding failure (i.e. $l < 2n$ before the deadline).

TABLE 5 – Implementation results for ECC processors with recoding methods from the literature ($\star$ means estimated values).

Work	area slices (FF/LUT)	BRAM	DSP	freq. MHz	dura. ms	FPGA	SCA prot.
[10]	4655 (4876/5740)	11	37	250	0.44	Virtex-4	Y
	1725 (4792/4177)	10	37	291	0.38	Virtex-5	Y
	256 b, generic curve, randomized window recoding						
[4]	6203 ALMs	0	92	160	0.44	Stratix-II	Y
	192 b, generic curve, Montgomery Ladder						
[3]	1793 (3641/6182)	6	0	155	7.6$\star$	Virtex-5	N
	2427 (4297/6981)	6	0	142	5.8$\star$	Virtex-5	N
	160 b, generic curve, MBNS recoding						
[1]	1828 (n.a./n.a.)	6$\star$	0	24	22.1	Virtex-5	Y
	2049 (n.a./n.a.)	6$\star$	0	24	15.5	Virtex-5	Y
	256 b, SECP curve, Montgomery Ladder						

and NAF-3, while providing natural protection against simple power analysis. It remains to be seen whether the addressing signal used to select the steps (big or small) may leak in the context of a Differential Power Analysis (DPA).

Acknowledgment

This work has been supported in part by the PAVOIS project (ANR 12 BS02 002 01).

Bibliographie

1. Baldwin (B.), Goundar (R. R.), Hamilton (M.) et Marnane (W. P.). – Co-Z ECC scalar multiplications for hardware, software and hardware-software co-design on embedded systems. *Journal of Cryptographic Engineering*, vol. 2, n. 4, novembre 2012, pp. 221–240.
2. Brier (E.), Joye (M.) et Dechene (I.). – *Embedded Cryptographic Hardware*, chap. Unified Point Addition Formulae for Elliptic Curve Cryptosystems, pp. 247–256. – Nova Science, 2004.
3. Chabrier (T.) et Tisserand (A.). – On-the-fly multi-base recoding for ECC scalar multiplication without pre-computations. – In *Proc. Symposium on Computer Arithmetic (ARITH)*, pp. 219–228. IEEE, avril 2013.
4. Guillermin (N.). – A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$. – In *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, *LNCS*, volume 6225, pp. 48–64. Springer, août 2010.
5. Hankerson (D.), Menezes (A.) et Vanstone (S.). – *Guide to Elliptic Curve Cryptography*. – Springer, 2004.
6. Herbaut (F.), Liardet (P.-Y.), Meloni (N.), Teglia (Y.) et Veron (P.). – Random euclidean addition chain generation and its application to point multiplication. – In *Proc. 11th International Conference on Progress in Cryptology (INDOCRYPT)*, *LNCS*, volume 6498, pp. 238–261. Springer, décembre 2010.
7. Herbaut (F.) et Veron (P.). – A public key cryptosystem based upon euclidean addition chains. – In *Proc. 6th International Conference on Sequences and Their Applications (SETA)*, *LNCS*, volume 6338, pp. 284–297. Springer, septembre 2010.
8. Joye (M.). – *Advances in Elliptic Curve Cryptography*, chap. Defenses Against Side-Channel Analysis, pp. 87–100. – Cambridge University Press, avril 2005, *London Mathematical Society Lecture Note*, volume 317.
9. Knuth (D. E.). – *Seminumerical Algorithms*. – Addison-Wesley, 1997, 3rd édition, *The Art of Computer Programming*, volume 2.
10. Ma (Y.), Liu (Z.), Pan (W.) et Jing (J.). – A high-speed elliptic curve cryptographic processor for generic curves over GF(p). – In *Proc. 20th International Workshop on Selected Areas in Cryptography (SAC)*, *LNCS*, volume 8282, pp. 421–437. Springer, août 2013.
11. Mangard (S.), Oswald (E.) et Popp (T.). – *Power Analysis Attacks : Revealing the Secrets of Smart Cards*. – Springer, 2007.
12. Meloni (N.). – New point addition formulae for ECC applications. – In *Proc. 1st International Workshop on Arithmetic of Finite Fields (WAIFI)*, *LNCS*, volume 4547, pp. 189–201. Springer, juin 2007.
13. Montgomery (P. L.). – Speeding the pollar and elliptic curves methods of factorisation. *Mathematics of Computation*, vol. 48, n. 177, janvier 1987, pp. 243–264.
14. Oswald (E.). – *Advances in Elliptic Curve Cryptography*, chap. Side Channel Analysis, pp. 69–86. – Cambridge University Press, avril 2005, *London Mathematical Society Lecture Note Series*, volume 317.