Error detection in addition chain based ECC point multiplication

S. Pontarelli[◇⋆], G.C. Cardarilli[◇], M. Re[◇], A. Salsano[◇]
{pontarelli, salsano}@ing.uniroma2.it, {marco.re, g.cardarilli}@ieee.org
[◇] University of Rome "Tor Vergata", Via del Politecnico 1, 00191, Rome, ITALY
[⋆] (ASI) Italian Space Agency, Viale Liegi, 26, 00198 Rome, ITALY

***Abstract*— In this paper the problem of error detection in elliptic curve point multiplication is faced. Elliptic Curve Point Multiplication is often used to design cryptographic algorithms that use fewer bits than other methods with the same security level. One of the mode used to break the security of cryptosystem is the injection of a fault in the hardware realizing the cryptographic algorithm. Therefore, to avoid this kind of attack, is very important to develop cryptosystems that are able to detect errors induced by a fault. The paper takes into account the algorithm for elliptic Curve Point Multiplication based on a sequence of additions called "addition chain" and shows how suitable modifications of the algorithms used for computing the point multiplication adds the error detection property to the algorithm.**

I. Introduction

Elliptic Curve Point Multiplication (ECC) is often used to design cryptographic algorithms that use fewer bits than other methods with the same security level. Because the cryptanalysis applied to the Elliptic Curve Cryptography (ECC) is unfeasible, a realistic attack to a cryptosystem must take into account the information extracted from the physical implementation of the cryptosystem. The injection of fault can be used to discover the secret key, as originally proposed in [1] and [2].

To face the use of fault attack, ECC systems with self-checking or error detection properties have been proposed. The work in [3] is on an ECC based on a finite field $\mathbb{F}_p$ and use parity-preserving circuits [4] to detect errors inside the modular operations on $\mathbb{F}_p$ implemented by using a redundant binary encoding. Instead, [5] and [6] face this problem for a finite field of type $\mathbb{F}_{2^n}$. All these methods are focused on the detection of faults in the single operation performed on the finite field. Instead, our method does not operate at the level of the operation on the finite field, but at level of the addition operation between points on the elliptic curve. This characteristic of our method allows implementing its in both hardware and software with very few modifications. The algorithm taken into account is based on the Euclidean Addition Chains (EAC) and has been proposed in [7] to compute the elliptic curve point multiplication. This algorithm is proposed in [7] to avoiding a side channel attack based on the power consumption analysis. Our modification provides also robustness against fault attacks.

II. ECC Background

An elliptic curve over a finite field $\mathbb{F}_p$ (where p is a large prime number) is formed by the set of points (x, y) satisfying the Weierstrass equation

$$E : y^2 = x^3 + ax + b \tag{1}$$

with x, y, a and $b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \not\equiv 0$ (mod p). Adding the point at infinity ∞ the set E forms an additive group with ∞ as the neutral element and the opposite of P_1 is $-P_1 = (x_1, -y_1)$.

Given two points on the curve $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ the addition between P_1 and P_2 is the point $P_3 = (x_3, y_3)$, with x_3 and y_3 defined by the following equations:

$$x_3 = \lambda^2 - (x_1 + x_2), \qquad y_3 = \lambda(x_1 - x_3) - y_1 \tag{2}$$

$$\text{where } \lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, \text{ if } P_1 \neq \pm P_2 \\ \dfrac{3x_1^2 + a}{2y_1}, \text{ if } P_1 = P_2 \end{cases}$$

The point scalar multiplication is defined as:

$$E \times \mathbb{Z} \ , \ \rightarrow E(P, k) \rightarrow Q = [k]P = \underbrace{P + P + \cdots + P + P}_{k \ times} \tag{3}$$

III. Point Multiplication by means of Euclidean Addition Chain

One of the methods proposed to compute the multiplication presented in equation (3) is based on the Euclidean Addition Chain [7]. An introduction to this method to speed up multiplications is given by Knuth in [8]. An addition chain for k is a list of positive integers

$$a_1 = 1, a_2, \ldots, a_l = k,$$

such that for each $i > 1$, there is some i_1 and i_2 with $1 \leq i_1 \leq i_2 < i$ and $a_i = a_{i_1} + a_{i_2}$. A short addition chain for k gives a fast algorithm for computing $[k]P$ by computing a_2P, a_3P, $\ldots$ a_lP. A simplification of the algorithm that use the addition chain can be done restricting the value that can assume i_1 and i_2 for each i. The euclidean restriction fixes the index $i_1 = i - 1$ and the index $i_2 = i - 2$ or $i_2 = j$, where

978-1-4244-4595-0/09/$25.00

j is the value of i_2 in the previous iteration. Therefore, the definition of the Euclidean addition chain (EAC) is as follows:

given the sequence $(a_1 = 1, a_2 = 2, \ldots, a_l = k)$ $(\forall i > 2)$ $a_i = a_{i-1} + a_{i-2}$ (big step) or $a_i = a_{i-1} + a_j$ (small step), where j satisfies the following relation: $a_{i-1} = a_{i-2} + a_j$.
If the value of a_i is the sum of the two biggest available numbers the sum is called a "big step", otherwise the sum is called a "small step". An example of an EAC is the following sequence: (1, 2, 3, 4, 7, 11, 18, 29, 40). Therefore, to compute $Q = 40P$ we compute the following seven additions:

Example 1: computing $Q = 40P$

1: $3P \leftarrow 2P + P$
2: $4P \leftarrow 3P + P$
3: $7P \leftarrow 4P + 3P$
4: $11P \leftarrow 7P + 4P$
5: $18P \leftarrow 11P + 7P$
6: $29P \leftarrow 18P + 11P$
7: $40P \leftarrow 29P + 11P$

For an EAC we can define a binary sequence $c = (c_1, c_2, \ldots, c_l)$ in which $c_i = 1$ if a_i is a big step, 0 if a_i is a small step. Now, we present the algorithm for point multiplication by using an EAC.

Algorithm 1: Point multiplication with EAC

Input: P, $[2]P$, and the binary sequence representing the EAC $(1, 2, a_3, a_4, \ldots a_l = k)$
Output: $Q = [k]P$.

1: $U_1 \leftarrow 2[P]$, $U_2 \leftarrow P$
2: **for** $i = 3$ **to** l **do**
3: $\quad U_1 \leftarrow U_1 + U_2$
4: $\quad$ **if** $c_i = 1$ **then**
5: $\quad\quad U_2 \leftarrow U_1$
6: $\quad$ **else**
7: $\quad\quad U_2 \leftarrow U_2$
8: $\quad$ **end if**
9: **end for**
10: **return** U_1

IV. Error detection in the Euclidean Addition Chain Based Multiplication

Now, we describe how to modify the EAC Based Multiplication to detect an error in the computation of the intermediate results. The method is composed by two parts:

1) a modification of the algorithm that computes also the difference between two points on the curve,
2) an algorithm that computes both the addition and the subtraction of two points on the curve.

Therefore, this section is divided in two parts. In the former we describe the modification of the algorithm for EAC based multiplication, in the latter we show the addition/subtraction procedure.

A. Modified Algorithm

To explain how our method works we see in detail how the values of the EAC are stored and discarded into the variables U_1 and U_2. If at the step i the algorithm performs a big step, the results of $U_1(i)$ and $U_2(i)$ are:

$$U_1(i) = U_1(i-1) + U_2(i-1)$$
$$U_2(i) = U_1(i-1)$$

and the value stored in $U_2(i-1)$ before the execution of line 4 of the algorithm is discarded, while the value stored in $U_1(i-1)$ is saved. Instead, if the first step is a small step is discarded. The old value stored in $U_1(i-1)$ and is preserved the value in $U_2(i-1)$. The difference between $U_1(i)$ and $U_2(i)$ is:

$$U_D(i) = U_1(i) - U_2(i) = U_1(i-1) + U_2(i-1) - U_2(i) \quad (4)$$

Because $U_2(i)$ is one of the two value $U_1(i-1)$ or $U_2(i-1)$ the difference U_D can be only the other value of the couple $U_2(i-1)$ and $U_1(i-1)$. Therefore, the difference between $U_1(i)$ or $U_2(i)$ can be used to check if an error is occurred in the algorithm. We modify Algorithm 1 computing also the difference U_D and storing as a check value the value of U_1 or U_2 that is discarded in the original algorithm. Moreover, we add two addition/subtraction steps to check also the last two values of the addition chain. The algorithm is presented in Algorithm 2.

Also the modified algorithm presents the same "power balanced" behavior independently by the value of the elements of the binary sequence c representing the EAC, and therefore presents the same robustness with respect to a SPA attack[7]. We applied the modified algorithm to the addition chain of example 1 to show how out method works. The results are presented in Table I, in which, for each step of algorithm 2 the values of U_1, U_2, U_D and U_C are shown.

step	U_1	U_2	U_D	U_C
1	$2P$	P	–	P
2	$3P$	P	P	$2P$
3	$4P$	$3P$	$2P$	P
4	$7P$	$4P$	P	$3P$
5	$11P$	$7P$	$3P$	$4P$
6	$18P$	$11P$	$4P$	$7P$
7	$29P$	$11P$	$7P$	$18P$
8	$40P$	$11P$	$18P$	$29P$
9	$51P$	$11P$	$29P$	$40P$
10	$62P$	$11P$	$40P$	$51P$

TABLE I
Computing $Q = 40P$ with error detection

The last two rows of the table are used only to check if the two last values ($29P$ and $40P$) are corrects. The overhead of the algorithm presented here is due to:

1) the computation of U_D, the storing of U_C and the equality checking of U_C and U_D.
2) the two additional steps performed at the end of the loop.

We notice that for a 160-bit encryption usually the EAC have a length of 320 bits or more. Therefore, the two steps performed

Algorithm 2: Modified Point multiplication with EAC

Input: P, $[2]P$, and the binary sequence representing the EAC $(1, 2, a_3, a_4, \dots a_l = k)$
Output: $Q = [k]P$, Error detection.

1: $U_1 \leftarrow 2[P]$, $U_2 \leftarrow P$, $U_C \leftarrow P$
2: **for** $i = 3$ **to** l **do**
3: $U_1 \leftarrow U_1 + U_2$, $U_D \leftarrow U_1 - U_2$
4: **if** $U_D \neq U_C$ **then**
5: **return** $Q = 0$, error=1
6: **end if**
7: **if** $c_i = 1$ **then**
8: $U_2 \leftarrow U_1$, $U_C \leftarrow U_2$
9: **else**
10: $U_2 \leftarrow U_2$, $U_C \leftarrow U_1$
11: **end if**
12: **end for**
13: $Q \leftarrow U_1$
{the following steps check the last two results of the EAC}
14: $U_1 \leftarrow U_1 + U_2$, $U_D \leftarrow U_1 - U_2$
15: **if** $U_D \neq U_C$ **then**
16: **return** $Q = 0$, error=1
17: **end if**
18: $U_2 \leftarrow U_2$, $U_C \leftarrow U_1$, $U_1 \leftarrow U_1 + U_2$, $U_D \leftarrow U_1 - U_2$
19: **if** $U_D \neq U_C$ **then**
20: **return** $Q = 0$, error=1
21: **end if**
22: $U_2 \leftarrow U_2$, $U_C \leftarrow U_1$
23: **return** Q, error=0

at the end of the loop have a negligible impact on the algorithm performance. Instead, the computation of U_D can be very time-consuming. In the next subsection, we will show that the computation of U_C and U_D can be done with a few efforts.

B. Addition/Subtraction in Elliptic Curve over $\mathbb{F}_p$

The addition formula for two points P_1 and P_2 on the curve is presented in equation (2). The computation of the result requires one field inversion (**I**), four field multiplications (3**M**) and four additions (5**A**). The overall computation of the point addition requires **I**+3**M**+5**A**. The inversion operation is the more expensive task that must be performed to add two points. Now, the subtraction of two points P_1 and P_2 can be performed as the sum of P_1 with the opposite $(-P_2)$ of P_2. If the curve is defined over $\mathbb{F}_p$, the opposite of $P_2 = (x_2, y_2)$ is simply $-P_2 = (x_2, -y_2)$. The computation of $(x_2 - x_1)^{-1}$, needed to compute λ is therefore the same used for the addition of the two point. Performing both the addition and subtraction of the two points we can share the value of $(x_2 - x_1)^{-1}$, (and the value of $x_1 + x_2$) saving one inversion (and one addition) in the count of needed operations. The count of operations for computing $P1 \pm P_2$ is **I**+6**M**+7**A**.

V. Fault effects and error detection in the modified point multiplication with EAC algorithm

To describe how a fault can modifies the behavior of the presented algorithm we suppose that a fault can change the result of an elementary operation performed on the ground field $\mathbb{F}_p$ in an arbitrary way. The choice of this "arithmetic level" fault models allows an abstraction from the implementation of the algorithm and describes in a more general way how our modified algorithm works.

The first error we take into account is an error providing as output a value that is not a point on the curve E. Our method detects this kind of error. In fact, if a point that is not on the curve E is "added" to a point out of the curve the result is out of the curve. Therefore, when it is compared with a previous value the equality does not hold, and the error is detected.

Now, let us suppose that an error changes the result of an operation from a point $P \in E$ to another point $P' \in E$. If this error occurs in computing $U_D = U_1 - U_2$ (line 6 of Algorithm 3) it is immediately detected by line 7. Instead, if the error affects line 5 ($U_1 = U_1 + U_2$ is changed in $\tilde{U_1} = U_1 + U_2 + e$) at the next iteration step U_D is computed as $U_D = \tilde{U_1} - U_2 = U_1 + U_2 + e - U_2 = U_1 + e \neq U_C$ and it is detected in line 7.

VI. Conclusions

This paper discusses how to modify the Euclidean Addition Chain algorithm used for point multiplication in Elliptic Curve Cryptography to obtain the detection of errors in the algorithm computation. The modification introduced in the algorithm do not limit its characteristics in terms of balanced power consumption and therefore maintains its robustness with respect to a SPA attack. The introduced modifications require a computational overhead that is negligible with respect to the computation time of the original point multiplication algorithm.

References

[1] D. Boneh, R.A. DeMillo, and R.J. Lipton, "On the importance of checking cryptographic protocols for faults," *Lecture Notes in Computer Science*, vol. 1233, pp. 37–51, 1997.

[2] Eli Biham and Adi Shamir, "Differential fault analysis of secret key cryptosystems," *Lecture Notes in Computer Science*, vol. 1294, pp. 513–525, 1997.

[3] J. Francq, J.B. Rigaud, P. Manet, A. Tria, and A. Tisserand, "Error Detection for Borrow-Save Adders Dedicated to ECC Unit," in *Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on*, 2008, pp. 77–86.

[4] B. Parhami, "Fault tolerant reversible circuits," in *Proc. 40 thAsilomar Conf. Signals, Systems, and Computers, October*, 2006, pp. 1726–1729.

[5] W. Chelton and M. Benaissa, "Concurrent error detection in GF (2/sup m/) multiplication and its application in elliptic curve cryptography," *Circuits, Devices & Systems, IET*, vol. 2, no. 3, pp. 289–297, 2008.

[6] R. Stern, N. Joshi, K. Wu, and R. Karri, "Register Transfer Level Concurrent Error Detection in Elliptic Curve Crypto Implementations," in *Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007. Workshop on*, 2007, pp. 112–119.

[7] A. Byrne, F. Crowe, W.P. Marnane, N. Meloni, A. Tisserand, and E. Popovici, "SPA resistant elliptic curve cryptosystem using addition chains," *International Journal of High Performance Systems Architecture*, vol. 1, no. 2, pp. 133–142, 2007.

[8] D.E. Knuth, *The art of computer programming. Vol. 2: Seminumerical algorithms*, Addison-Wesley, Reading, Massachusetts, second edition, 1981.