LOEDAR: A Low Cost Error Detection and Recovery Scheme for ECC

Kun Ma
Department of Electric and Computer Engineering
University of Illinois at Chicago
IL, USA

Kaijie Wu
Department of Electric and Computer Engineering
University of Illinois at Chicago
IL, USA

***Abstract*—This paper presents LOEDAR, a novel low-cost Error Detection and Recovery scheme, for Montgomery Ladder Algorithm based Elliptic Curve Scalar Multiplication (ECSM). The LOEDAR scheme exploits the invariance among the intermediate results produced by the algorithm to detect errors. The error detection process can be carried periodically during ECSM to verify data correctness, and will recover the cryptosystem back to the latest checkpoint upon detecting errors. The frequency of running the error detection process can be adjusted to trade off the power and time overhead with error detection latency and recovery overhead. The hardware and power overhead of LOEDAR are about 37% and 69% respectively. Each additional error detection process contributes less than 1% additional time overhead and power overhead.**

Keywords – elliptic curve cryptography(ECC); elliptic curve scalar multiplication(ECSM); concurrent error detection; montgomery ladder;

I. INTRODUCTION

Elliptic Curve Cryptography (ECC), which was proposed by Neal Koblitz and Victor Miller, is an attractive approach to public-key cryptography. Compared with other public-key cryptosystems such as RSA, ECC provides the same level of security with smaller key size and other parameters. In ECC-based cryptosystems, the most time-consuming operation is the Elliptic Curve Scalar Multiplication (ECSM). ECSM takes inputs a point P on an elliptic curve E and a positive integer k and computes $Q = kP = P + P + \cdots + P$ (k times). P is called the base point and often made public, while k is the secret. ECSM is computed using two basic operations: point addition and point doubling. Point addition adds two different points, while point doubling adds a point to itself. Since both operations are defined geometrically on the elliptic curve E, they involve computations of coordinates of points. A number of ECSM algorithms have been proposed, and the most attractive one is Montgomery Ladder Algorithm [1]. An efficient implementation of this algorithm is presented in [2]. Montgomery Ladder Algorithm is based on the observation that the x-coordinate of the sum of two points can be computed using just the x-coordinates of the two points and the difference of the two points. Hence only x-coordinates are involved during the accumulation of P, and the y-coordinate is only involved at the last step, i.e. recovering the y-coordinate of the final result kP from its x-coordinate. In addition to cost reduction, Montgomery Ladder Algorithm also shows strong resistance to sign change attack [3].

Since ECC is commonly used for key exchange and digital signature generation, the reliability of ECC-based cryptosystems is especially important for secure communications. Faults caused by natural or artificial reasons may result in data corruption and, even worse, security leakage of the whole system. By analyzing the erroneous outputs due to the attacker-induced faults, the key information can be recovered with an affordable effort. A variety of such attacks, called fault attacks, to ECC-based cryptosystems have been presented in the past years [4] [5] [6] [7]. Since the point produced by a faulty calculation may leave the original curve with a high probability, an extra step, called Point Verification (PV) that checks if kP is still on the original curve before sending it out, has been recommended by SECG [8]. However, using PV as the only protection could be risky. First, PV misses some faults [9]. Second, attacks have been proposed to manipulate the faulty results so they can pass PV process [7]. Besides, PV only detects faults, and fault detection only may not be sufficient for the cryptosystems that need to perform operations properly even in the events of natural or attacker-induced faults. Such systems demand efficient fault recovery. The recovery following fault detection using PV, however, will have to start from the very beginning and re-compute the complete ECSM. Hence additional fault detection mechanism must be implemented to detect faults occurred during ECSM.

Most fault tolerant structures are based on parallel computation [10]. The structures have two or more identical and independent ECSM modules working in parallel, and faults are detected or corrected by comparing or voting the results from all parallel modules. Consequently, more than 100% hardware overhead is required. It is unacceptable for the embedded cryptosystems that have highly constrained area budget. Another easy option is to perform the ECSM operation twice or more times on the same datapath. The delay cost by recomputation will slow down system performance by at least 100%. Furthermore, in the view of power consumption, both approaches are always taking twice or more energy to finish a regular ECSM computation. This is unaffordable for the systems operating on limited energy supply. In this paper, we present a novel low-cost error detection and recovery (LOEDAR) scheme for the Montgomery Ladder Algorithm. Besides its low area and time overhead, the most prominent benefit of LOEDAR is that one can easily adjust the trade-off between energy and time consumed by error detection and the error detection latency and recovery cost. A smaller error detection latency results in a faster recovery when faults are

This work is supported by NSF grant CNS-0831301.
978-3-9810801-7-9/DATE11/©2011 EDAA

present, but incur more energy consumption when faults are absent.

The organization of the paper is as follows. Section 2 presents a brief overview of ECC and Montgomery Ladder Algorithm. Section 3 presents the proposed LOEDAR scheme. Its error detection capability is analyzed in Section 4. The experimental results are shown in Section 5. Section 6 concludes this paper.

II. ECC AND MONTGOMERY LADDER ALGORITHM

A. ECC Overview

ECC can be implemented over two major types of Galois finite fields: prime finite field GF(p) and binary finite field $\mathrm{GF}(2^m)$. Our work focuses on the $\mathrm{GF}(2^m)$ field due to its hardware-efficient field arithmetic operations. An elliptic curve E over $\mathrm{GF}(2^m)$ is defined to be a set of points $(x, y) \in \mathrm{GF}(2^m) \times \mathrm{GF}(2^m)$ that satisfy the following equation where a and $b \in \mathrm{GF}(2^m)$, $b \neq 0$, and the point at infinity $\mathcal{O}$:

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

Based on the geometric property of elliptic curve, two basic functions, point addition and point doubling, are defined on the elliptic curve. Both functions involve a number of arithmetic operations over $\mathrm{GF}(2^m)$ including multiplication, square, addition and division. Let $P_1 = (x_1, y_1) \in E$, $P_2 = (x_2, y_2) \in E$, and $P_1 \neq \pm P_2$. Point addition $P_1 + P_2$ will result in a new point $P_3 = (x_3, y_3)$ on elliptic curve , where

$$\begin{cases} x_3 = \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)^2 + \dfrac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a \\ y_3 = \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1 \end{cases} \tag{2}$$

If $P_1 = P_2$, P_3 will be equal to $2P_1$ and the operation is referred to as point doubling, and the coordinates are computed as follows.

$$\begin{cases} x_3 = x_1^2 + \dfrac{b}{x_1^2} \\ y_3 = x_1^2 + \left(x_1 + \dfrac{y_1}{x_1}\right)x_3' + x_3' \end{cases} \tag{3}$$

B. Montgomery Ladder Algorithm

In ECC, the major computation is ECSM. A number of ECSM algorithms have been proposed, among which Montgomery Ladder Algorithm is the most attractive one due to its efficient implementation [2] and the capability to resist some fault attacks such as sign change attack [3]. Montgomery Ladder Algorithm is given in Algorithm 1. The l-bit secret k is represented using its binary form $k = (k_0 k_1 \ldots k_{l-1})_2$, where k_0 is the MSB and k_{l-1} is the LSB. It is assumed that $k_0 = 1$.

Algorithm 1. Montgomery Ladder Algorithm
Input: an integer $k \geq 0$, and a point $P = (x, y) \in E$
Output: $Q = kP$.
1. Set $k \leftarrow (k_0 k_1 \ldots k_{l-1})_2$
2. Set $P_1 \leftarrow P$, $P_2 \leftarrow 2P$
3. for i from 1 to $l - 1$ do
    if $k_i = 1$ then $P_1 \leftarrow P_1 + P_2$, $P_2 \leftarrow 2P_2$.
    else $P_2 \leftarrow P_2 + P_1$, $P_1 \leftarrow 2P_1$.
4. return ($Q = P_1$).

As one may notice that in this algorithm, the difference between P_2 and P_1 is always P (i.e. $P_2 - P_1 = P$). This property helps reduce the complexity of computation significantly. Instead of applying (2), the x-coordinate of the sum $P_1 + P_2$ can be computed in terms of the x-coordinates of P_1, P_2 and their difference P.

$$x(P_1 + P_2) = x + \left(\frac{x_1}{x_1 + x_2}\right)^2 + \frac{x_1}{x_1 + x_2} \tag{4}$$

where x, x_1 and x_2 are the x-coordinate of P, P_1 and P_2, respectively. Hence both point addition and point doubling in Algorithm 1 involve only x-coordinates. After finishing the main loop, the y-coordinate of $Q = kP$ can be recovered by using the coordinates of P, P_1, and P_2 of the last iteration.

Because division in $\mathrm{GF}(2^m)$ is costly and time-consuming, projective coordinates have been suggested to reduce the number of times division is performed. The affine coordinates of a point $P = (x, y)$ is transformed to projective coordinates $P = (X, Y, Z)$ with $x = X/Z$ and $y = Y/Z$. Point addition and point doubling in projective coordinates are given in (5) and (6), respectively. Hence division is only performed at the last step which transforms the projective coordinates of the points to affine coordinates for output.

$$\begin{cases} Z(P_1 + P_2) = Z_3 = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2 \\ X(P_1 + P_2) = X_3 = x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1) \end{cases} \tag{5}$$

$$\begin{cases} X(2P_1) = X_4 = X_1^4 + b \cdot Z_1^4 \\ Z(2P_1) = Z_4 = X_1^2 \cdot Z_1^2 \end{cases} \tag{6}$$

III. THE LOEDAR SCHEME

The point-updating pyramid shown in Figure 1 illustrates how points P_1 and P_2 are updated in the first three iterations in the *for* loop of Algorithm 1 that traverses from the MSB k_1 to the LSB k_{l-1} . Please ignore P_V for now. At iteration *i*, the ECSM process will take either the left path (if $k_i = 0$) or the right path (if $k_i = 1$). This is repeated until the leaf k_{l-1} is reached. For example, the path shown in bold arrows is taken when $k = 11 = (k_0 k_1 k_2 k_3)_2 = (1011)_2$, and the results after the 3rd iteration are $P_1 = 11P$, and $P_2 = 12P$.

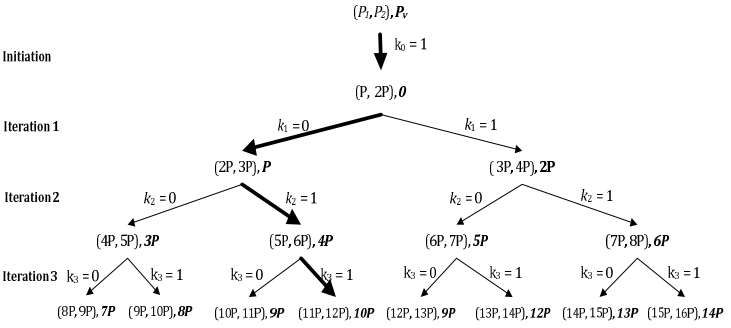

Figure 1. Montgomery Ladder Algorithm point updating graph

A. The Idea

A good property of the algorithm is that the difference between P_2 and P_1 is always kept as P if no fault occurs. Our first intuitive is to verify if the constant difference is kept at the end of each iteration, i.e. checks if $P_2 = P_1 + P$. This idea, however, does not work well because of the following: If this verification is to be done in the original affine coordinates, one will need to first recover the y-coordinates of all the three points and then perform the operations specified in (2), which is too time-consuming; If this verification is to be done in the projective coordinates, one may need to know the x-coordinate of the difference between P_1 and P. However, it is hardly

possible to know $P_1 - P$ without adding extra computation since P_1 changes in every iteration.

In order to take the advantage of computing point addition and point doubling in projective coordinates, we introduce a new point called Verification Point, P_v, as shown in Figure 1. P_v is initialized to $\mathcal{O}$, and is incremented at each iteration just like P_1 and P_2. Let us denote by P_1^i, P_2^i, P_v^i the value of points P_1, P_2, P_v after the i^{th} iteration, respectively. During the i^{th} iteration P_v is incremented by either P_1 or P_2 depending on k_i, i.e. $P_v^i = P_v^{i-1} + P_1^{i-1}$ if $k_i = 0$, or $P_v^i = P_v^{i-1} + P_2^{i-1}$ if $k_i = 1$. In the example given in Figure 1, $P_v^1 = P_v^0 + P_1^0 = P$ since $k_1 = 0$; $P_v^2 = P_v^1 + P_2^1 = 4P$ since $k_2 = 0$, and $P_v^3 = P_v^2 + P_2^2 = 10P$ since $k_3 = 1$. Lemma 1 shows that after each iteration $P_v^i + P_2^i = 2P_1^i$ always holds if no fault occurs.

Lemma 1: If P_v is updated as described above, then $P_v^i + P_2^i = 2P_1^i$ for $1 \le i \le l$-1 if no fault occurs.

Proof. Before Algorithm 1 starts, P_1^0 , P_2^0 , and P_v^0 are initialized to P, $2P$, and $\mathcal{O}$ respectively where P is the base point on the elliptic curve. After the 1st iteration, P_1, P_2 and P_v are updated as follows.

$$P_1^1 = \begin{cases} 2P & when\ k_1 = 0, \\ 3P & when\ k_1 = 1. \end{cases}$$

$$P_2^1 = \begin{cases} 3P & when\ k_1 = 0, \\ 4P & when\ k_1 = 1. \end{cases}$$

$$P_v^1 = \begin{cases} \mathcal{O} + P = P & when\ k_1 = 0, \\ \mathcal{O} + 2P = 2P & when\ k_1 = 1. \end{cases}$$

Thus $P_v^1 + P_2^1 = 2P_1^1$ holds for i = 1 no matter $k_1 = 0$ or 1.

Now assume that $P_v^q + P_2^q = 2P_1^q$ holds for $i = q \ge 1$, and we will show that $P_v^{q+1} + P_2^{q+1} = 2P_1^{q+1}$ always holds:

$$P_1^{q+1} = \begin{cases} 2P_1^q & when\ k_{q+1} = 0, \\ P_1^q + P_2^q & when\ k_{q+1} = 1. \end{cases}$$

$$P_2^{q+1} = \begin{cases} P_1^q + P_2^q & when\ k_{q+1} = 0, \\ 2P_2^q & when\ k_{q+1} = 1. \end{cases}$$

and verification point

$$P_v^{q+1} = \begin{cases} P_v^q + P_1^q & when\ k_{q+1} = 0, \\ P_v^q + P_2^q & when\ k_{q+1} = 1. \end{cases}$$

When $k_{q+1} = 0$, we have $P_v^{q+1} + P_2^{q+1} = P_v^q + P_1^q + P_1^q + P_2^q = 4P_1^q = 2P_1^{q+1}$. When $k_{q+1} = 1$, we have $P_v^{q+1} + P_2^{q+1} = P_v^q + P_2^q + 2P_2^q = 2\left(P_1^q + P_2^q\right) = 2P_1^{q+1}$. Thus, $P_v^{q+1} + P_2^{q+1} = 2P_1^{q+1}$ holds. This proves Lemma 1.

Recall that with the knowledge of the difference between P_1 and P_2, $P_1 + P_2$ can be implemented efficiently where only the x-coordinates of points are involved [2]. For the same reason, incrementing P_v involves only the x-coordinates as well – the difference between P_v and the incremental value of P_v is known, which is either P if $k_i = 0$ or $2P$ if $k_i = 1$. Hence P_v can be computed efficiently in the same way as $P_1 + P_2$.

B. The LOEDAR Scheme

The proposed LOEDAR scheme is shown in Algorithm 2. All the operations are performed in projective coordinates. x, the affine x-coordinate of a point, will be transformed to projective coordinates (X, Z) where $X/Z = x$. Algorithm 2 has two parts. Part 1 updates P_1 and P_2 just like Algorithm 1, which will be referred to as the normal scalar multiplication. Part 1 also updates P_v concurrently with P_1 and P_2, which will be referred to as P_v accumulation. Between the two iterations of Part 1, Part 2 verifies the correctness and triggers recovery if errors are detected, and it will be referred to as EDR. In EDR errors are detected by verifying the equivalence of $P_v + P_2$ and $2P_1$ at the end of an iteration, which involves a point addition that calculates the X and Z coordinates of $P_v + P_2$ (denoted by $X_{P_v+P_2}$ and $Z_{P_v+P_2}$), a point doubling that calculates the X and Z coordinates of $2P_1$ (denoted by X_{2P_1} and Z_{2P_1}), and two multiplications followed by a comparison to verify if $X_{P_v+P_2} \cdot Z_{2P_1} = Z_{P_v+P_2} \cdot X_{2P_1}$. The equation, if holds, implies that $X_{P_v+P_2}/Z_{P_v+P_2} = X_{2P_1}/Z_{2P_1}$ which further implies $P_v + P_2 = 2P_1$. We don't directly compare if $X_{P_v+P_2} = X_{2P_1}$ and $Z_{2P_1} = Z_{P_v+P_2}$ because transformation between affine coordinates and projective coordinates is not unique and different pairs of (X, Z) may map to the same affine coordinate x. If no error is detected, P_1, P_2, and P_v will be saved as a checked state and Part 1 is resumed. Otherwise Part 1 will rewind to the previous checked state and resume from there.

Algorithm 2. The LOEDAR Scheme
Input: An integer $k \ge 0$, and a point $P = (x, y) \in E$
Output: $Q = kP$

Part 1: Scalar Multiplication and P_v accumulation
1. Set $k \leftarrow (k_0 k_1 \dots k_{l-1})_2$.
2. $P_1 \leftarrow P$, $P_2 \leftarrow 2P$, $P_v \leftarrow \mathcal{O}$
3. for i from 1 to l-1 do
   if $k_i = 1$ then
   $\boldsymbol{P_v \leftarrow P_v + P_2}$,
   $P_1 \leftarrow P_1 + P_2$, $P_2 \leftarrow 2P_2$.
   else
   $\boldsymbol{P_v \leftarrow P_v + P_1}$,
   $P_2 \leftarrow P_2 + P_1$, $P_1 \leftarrow 2P_1$.
   end if
4. return ($Q = P_1$).

Part 2: EDR
Pause Part 1 at the end of an iteration;
if $P_v + P_2 = 2P_1$
   Save them as a new checked state; Resume Part 1;
else
   Retrieve the previous checked state; Resume Part 1;
end if

The corresponding LOEDAR architecture is shown in Figure 2. It is composed of an ECSM/EDR module, an accumulation module, a coordinate transformation module, a point verification module, and a register file. P_v is computed by Accumulation module and updated after every iteration during normal scalar multiplication. The ECSM/EDR module updates P_1 and P_2 during normal scalar multiplication and verifies if $P_v + P_2 = 2P_1$ during EDR. At the end of a certain iteration when an EDR process is about to run, the ECSM module suspends the normal scalar multiplication and turns to EDR. After finishing all the iterations of the scalar multiplication, the projective coordinates of P_1 are transformed to affine coordinates via coordinate transformation module, and the result will be output only after it passes point verification (PV)[1].

In our implementation, each iteration of scalar multiplication takes 17 cycles while each EDR process takes 21 cycles. If EDR process is carried after each iteration of scalar multiplication, the time overhead is worse than the

[1] Note that the coordinate transformation before scalar multiplication is trivial as one can simply set $X = x$ and $Z = 1$.

straightforward time-redundancy based techniques. However, the LOEDAR has an advantage that one can run the EDR process much less frequently, i.e. one EDR after every N iterations of scalar multiplication and N could be any number between 1 and l-1. The error detection capability is analyzed in the next section.

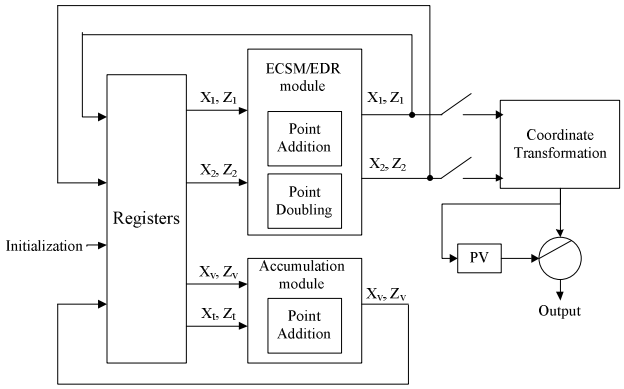

Figure 2. The Architecture for LOEDAR

IV. The Analysis of Fault Detection Capability

It is assumed that fault occurrences are a random event even though they are introduced deliberately. This indicates that any module could be affected by faults with a probability depending on its silicon area. The proposed scheme detects and recovers the faults occurred during scalar multiplication and ensures that point sent to the coordinate transformation is correct. Faults occurred during coordinate transformation, though will not leak the information of the secret, will be detected by point verification, and can be recovered by just transforming the coordinates one more time. The register file and the FSM controller (not shown in the figure) are subject to faults as well. While the scheme does not protect the register file directly, faults in most of these registers will be detected since they are no different from the faults occurred in the datapath. Only the ones that store the checked state require special attention. However, they can be protected efficiently using Error-Correcting Codes. The proposed scheme has a rather simple state machine, which can be made fault-tolerant by duplication/triplication with little overhead in the large, or using error detecting/correcting codes with minimum overhead as suggested in [11]. Hence the analysis will focus on the faults occurred in ECSM module and Accumulation module. Further, we assume that faults injected by attackers are transient – i.e. a fault affects only one operation. While multiple faults could occur as well, the possibility is small. Due to the page limit and the number of possible fault patterns, the following analysis assumes there is at most one fault between two successive EDR processes

A single transient fault can affect either an operation in scalar multiplication or an operation in P_v accumulation, or an operation in EDR. While the proposed scheme protects the operations in all the three functions, the capability of detecting the faults of scalar multiplication and P_v accumulation is of the most interest to us. This is because the major goal of this technique is to deliver correct P_1, P_2 and P_v . On the other hand, the capability of detecting the faults of EDR is of less interest since missing those faults won't affect the correctness of P_1, P_2 and P_v at all. Hence our analysis will focus on the former.

Figure 3 shows the Data Flow Graph (DFG) of the point addition and point doubling in scalar multiplication and the point addition in P_v accumulation. The operations of the point addition and point doubling in the scalar multiplication are numbered from A1 to A7 and from D1 to D7, respectively, while the operations of the point addition in the P_v accumulation are numbered from V1 to V7. Here ×, +, and S inside circles denote multiplier, adder and squarer over GF(2^m), respectively. For example, ×1 refers to multiplier 1 and is used four times (A1, A2, A3, A6) in the point addition of scalar multiplication. Each of the three functions is allocated one multiplier, one adder and one squarer exclusively, and no hardware is shared between them.

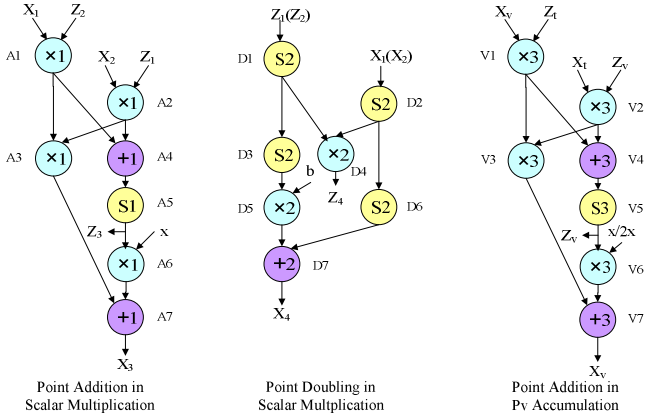

Figure 3. DFG of point addition and point doubling in scalar multiplication and P_v accumulation

We will first analyze the case where a fault occurs in the iteration right before an EDR process. The fault may occur in any operation. Let's take A1 as an example and assume the fault introduces an offset O to its output. The offset will be passed through all the subsequent operations and will eventually affect X_3 and Z_3, which are the coordinates of $P_3 = P_1 + P_2$. We hence denote them by X_3' and Z_3'.

$$Z_3' = [(X_1Z_2 + O) + X_2Z_1]^2 = Z_3 + O^2$$

$$X_3' = xZ_3 + (X_1Z_2 + O)X_2Z_1 = X_3 + xO^2 + OX_2Z_1$$

If $X_3'/Z_3' = X_3/Z_3$, (X_3', Z_3') is another mapping of P_3 in projective coordinates. This is because the mapping is not unique and a point in affine coordinates may have multiple mappings in projective coordinates as long as $X/Z = x$. Hence (X_3', Z_3') can be deemed as a correct result in spite of the fault. This type of faults has no effect on the correctness of result and can be safely ignored.

If $X_3'/Z_3' \neq X_3/Z_3$, (X_3', Z_3') is not a mapping of P_3. We call it an error of P_3. Without loss of generality, X_3' and Z_3' can be expressed as follows when a transient fault occurs in any arithmetic unit of point addition of scalar multiplication:

$$Z_3' = Z_3 + \mathrm{F_a}(O), \qquad X_3' = X_3 + \mathrm{F_b}(O)$$

where $\mathrm{F_a}(O)$ and $\mathrm{F_b}(O)$ are functions of offset O and are listed in TABLE I. Meanwhile the coordinates (X_4, Z_4) obtained from the point doubling of scalar multiplication and $(X_v\ Z_v)$ from P_v accumulation are not affected by this fault.

If $k_i = 0$, the updated P_1 and P_2 are $P_1^U = P_4$, $P_2^U = P_3$. The EDR process verifies if $P_2^U + P_v = 2P_1^U$, that is $P_3 + P_v = 2P_4$.

$$Z'_{P_3+P_v} = Z_{P_3+P_v} + (\mathrm{F_b}(O)Z_v + X_v\mathrm{F_a}(O))^2$$
$$X'_{P_3+P_v} = X_{P_3+P_v} + x\big(\mathrm{F_b}(O)Z_v + X_v\mathrm{F_a}(O)\big)^2 + X_vZ_v\big(X_3\mathrm{F_a}(O) + \mathrm{F_b}(O)Z_3 + \mathrm{F_a}(O)\mathrm{F_b}(O)\big)$$
$$Z'_{2P_4} = Z_{2P_4}$$
$$X'_{2P_4} = X_{2P_4}$$

If $Z'_{P_3+P_v}X'_{2P_4} = X'_{P_3+P_v}Z'_{2P_4}$, the fault will be missed. This happens only when Osatisfies the following equation:

$$X_{2P_4}(\mathrm{F_b}(O)Z_v + X_v\mathrm{F_a}(O))^2 = Z_{2P_4}\Big(x\big(\mathrm{F_b}(O)Z_v + X_v\mathrm{F_a}(O)\big)^2 + X_vZ_v\big(X_3\mathrm{F_a}(O) + \mathrm{F_b}(O)Z_3 + \mathrm{F_a}(O)\mathrm{F_b}(O)\big)\Big)$$

TABLE I. THE $\mathrm{F_a}(0)$ AND $\mathrm{F_b}(0)$ OF THE POINT ADDITION OF SCALAR MULTIPLICATION

Faulty operation	$\mathbf{F_a}(\boldsymbol{O})$	$\mathbf{F_b}(\boldsymbol{O})$
A1	O^2	$xO^2 + OX_2Z_1$
A2	O^2	$xO^2 + OX_1Z_2$
A3	0	O
A4	O^2	xO^2
A5	O	xO
A6	0	O
A7	0	O

Consider O as the argument, the above equation has at most four solutions if the fault happens in operation A1, A2 or A4. Assuming over $GF(2^m)$ O can be any value between 1 and 2^m with equivalent probability, the probability of undetected error satisfies

$$Pr_{\text{undetected error}} \le \frac{4}{2^{\mathrm{m}}} = \frac{1}{2^{\mathrm{m-2}}}$$

If the fault occurs in A3, A5, A6 or A7, the above equation has at most 2 solutions. So the probability of undetected error satisfies

$$Pr_{\text{undetected error}} \le \frac{2}{2^{\mathrm{m}}} = \frac{1}{2^{\mathrm{m-1}}}$$

If $k_i = 1$, the updated P_1 and P_2 are $P_1^U = P_3$, $P_2^U = P_4$. The EDR process verifies if $P_2^U + P_v = 2P_1^U$, that is $P_4 + P_v = 2P_3$.

$$Z'_{P_4+P_v} = Z_{P_4+P_v}$$
$$X'_{P_4+P_v} = X_{P_4+P_v}$$
$$Z'_{2P_3} = Z_{2P_3} + Z_3^2\big(\mathrm{F_b}(O)\big)^2 + X_3^2\big(\mathrm{F_a}(O)\big)^2 + \big(\mathrm{F_a}(O)\mathrm{F_b}(O)\big)^2$$
$$X'_{2P_3} = X_{2P_4} + \big(\mathrm{F_b}(O)\big)^4 + \mathrm{b}\big(\mathrm{F_a}(O)\big)^4$$

If $Z'_{P_4+P_v}X'_{2P_3} = X'_{P_4+P_v}Z'_{2P_3}$, the fault will be missed. This happens only when O satisfies the following equation:

$$Z_{P_4+P_v}\Big(\big(\mathrm{F_b}(O)\big)^4 + \mathrm{b}\big(\mathrm{F_a}(O)\big)^4\Big) = X_{P_4+P_v}\Big(Z_3^2\big(\mathrm{F_b}(O)\big)^2 + X_3^2\big(\mathrm{F_a}(O)\big)^2 + \big(\mathrm{F_a}(O)\mathrm{F_b}(O)\big)^2\Big)$$

If the fault occurs in A1, A2 or A4, the above equation has at most 8 solutions. So the probability of undetected error satisfies

$$Pr_{\text{undetected error}} \le \frac{8}{2^{\mathrm{m}}} = \frac{1}{2^{\mathrm{m-3}}}$$

If the fault occurs in A3, A5, A6 or A7, the above equation has at most 4 solutions. So the probability of undetected error satisfies

$$Pr_{\text{undetected error}} \le \frac{4}{2^{\mathrm{m}}} = \frac{1}{2^{\mathrm{m-2}}}$$

In the same way, we obtain the probability of undetected error when a fault occurs in point doubling in scalar multiplication or the point addition in P_v accumulation. They are reported in TABLE III. and TABLE IV. respectively.

TABLE II. THE $\mathrm{Pr}_{\text{undetected error}}$ OF THE POINT DOUBLING OF SCALAR MULTIPLICATION

Faulty operations	k_i	$\boldsymbol{Pr}_{\text{undetected error}}$
A1, A2, A4	0	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-2}}}$
	1	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-3}}}$
A3, A5, A6, A7	0	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-1}}}$
	1	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-2}}}$

TABLE III. THE $\mathrm{Pr}_{\text{undetected error}}$ OF THE POINT DOUBLING OF SCALAR MULTIPLICATION

Faulty operations	k_i	$\boldsymbol{Pr}_{\text{undetected error}}$
D1, D2	0	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-3}}}$
	1	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-2}}}$
D3, D4, D5, D6, D7	0	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-2}}}$
	1	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-1}}}$

TABLE IV. THE $\mathrm{Pr}_{\text{undetected error}}$ OF THE POINT ADDITION OF P_v ACCUMULATION

Faulty operations	k_i	$\boldsymbol{Pr}_{\text{undetected error}}$
V1, V2, V4	0	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-2}}}$
	1	
V3, V5, V6, V7	0	$Pr_{\text{undetected error}} \le \frac{1}{2^{\mathrm{m-1}}}$
	1	

Overall the maximum probability of undetected error is no larger than $1/2^{m-3}$. Since m is usually larger than 160 for security reasons, this number is quite small.

We then analyze the case where a fault affects the q^{th} iteration and the next EDR process is after the n^{th} iteration, $n \ge q$. Lemma 2 shows that under the single transient fault model one can detect the error after the n^{th} iteration.

Lemma 2: If in the q^{th} iteration, $P_v^q + P_2^q \ne 2P_1^q$, then in the n^{th} iteration, $n > q$, inequation $P_v^n + P_2^n \ne 2P_1^n$ is true if no fault occurs between the q^{th} iteration and the n^{th} iteration.

Proof. Given $P_v^q + P_2^q \ne 2P_1^q$ and let $n = q + 1$, we have

$$P_1^{q+1} = \begin{cases} 2P_1^q & \text{when } k_{q+1} = 0, \\ P_1^q + P_2^q & \text{when } k_{q+1} = 1. \end{cases}$$

$$P_2^{q+1} = \begin{cases} P_1^q + P_2^q & \text{when } k_{q+1} = 0, \\ 2P_2^q & \text{when } k_{q+1} = 1. \end{cases}$$

$$P_v^{q+1} = \begin{cases} P_v^q + P_1^q & \textit{when } k_{q+1} = 0, \\ P_v^q + P_2^q & \textit{when } k_{q+1} = 1. \end{cases}$$

This gives the following:

$$P_v^{q+1} + P_2^{q+1}$$

$$= \begin{cases} P_v^q + 2P_1^q + P_2^q \neq 4P_1^q & when\ k_{q+1} = 0, \\ P_v^q + P_2^q + 2P_2^q \neq 2P_1^q + 2P_2^q & when\ k_{q+1} = 1. \end{cases}$$

Thus, after the $(q+1)^{th}$ iteration, $P_v^{q+1} + P_2^{q+1} \neq 2P_1^{q+1}$.

Now we assume that $P_v^i + P_2^i \neq 2P_1^i$ holds for $n = i, i \geq q+1$, we will show that $P_v^{i+1} + P_2^{i+1} \neq 2P_1^{i+1}$ also holds. After the $(i+1)^{th}$ iteration, we have the following:

$$P_1^{i+1} = \begin{cases} 2P_1^i & \text{when } k_{i+1} = 0, \\ P_1^i + P_2^i & \text{when } k_{i+1} = 1. \end{cases}$$

$$P_2^{i+1} = \begin{cases} P_1^i + P_2^i & \text{when } k_{i+1} = 0, \\ 2P_2^i & \text{when } k_{i+1} = 1. \end{cases}$$

$$P_v^{i+1} = P_v^i + P_t^i = \begin{cases} P_v^i + P_1^i & when\ k_{i+1} = 0, \\ P_v^i + P_2^i & when\ k_{i+1} = 1. \end{cases}$$

This gives the following:

$$P_v^{i+1} + P_2^{i+1} = \begin{cases} P_v^i + P_1^i + P_1^i + P_2^i \neq 4P_1^i & when\ k_{i+1} = 0, \\ P_v^i + P_2^i + 2P_2^i \neq 2P_1^i + 2P_2^i & when\ k_{i+1} = 1. \end{cases}$$

Hence $P_v^{i+1} + P_2^{i+1} \neq 2P_1^{i+1}$. This proves Lemma 2.

V. THE EXPERIMENTS

We modeled the complete ECC algorithm and the proposed LOEDAR scheme using VHDL, and synthesized it into netlist using Synopsys Design Vision and TSMC 65nm library. The ECSM/EDR module has two multipliers, two squarers and two adders, and the Accumulation module has a multiplier, a squarer and an adder. All operations are over $GF(2^{163})$. The area and delay data of the hardware units are reported in TABLE V. The maximum system frequency is above 200MHz.

TABLE V. THE AREA AND DELAY DATA OF THE HARDWARE UNITS

Arithmetic Unit	Area (um^2)	Time (clock cycles)
Multiplier	40066	4
Squarer	804	1
Adder	587	1
Divider	12835	Depends on input

Compared with regular ECSM, the hardware overhead of the proposed scheme is resulted from the additional Accumulation module and the registers for intermediate storage in EDR process. As shown in TABLE VI. , the hardware overhead is much less than TMR, DMR_PV and PRC methods [10].

TABLE VII. illustrates the time overhead and the switching activities overhead of the LOEDAR scheme over the regular ECSM. The time overhead of LOEDAR scheme depends on how many times the EDR process is performed. Every EDR process contributes a small fixed time overhead of 0.68%. Hence the overall time overhead can be expressed as n×0.68% where n is the total number of times the EDR process is performed during ECSM. The power overhead is reported in the form of average switching activities. We used VSS to simulate the gate-level netlist and captured the average switching activities over dozens of different inputs. Due to the Accumulation module, the switching activities produced by LOEDAR are 69% on average more than the regular ECSM when no EDR process is performed, and each EDR process will increase the switching activities by 0.6% on average.

TABLE VI. HARDWARE OVERHEAD

Scheme	Area (um^2)	Hardware Overhead
Regular ECSM	220502	-
LOEDAR	303401	37.6%

TABLE VII. TIME OVERHEAD AND SWITCHING ACTIVITIES OVERHEAD

Scheme	Time (clock cycles)/ overhead	Average Switching Activities(×10^3)/overhead
Regular ECSM	3084	166408
LOEDAR w/o EDR	3084/0%	281127 /69%
Each EDR process	21/0.68%	1020 /0.6%

VI. CONCLUSION

In this paper, we have presented a low-cost error detection and recovery scheme (called LOEDAR) for Montgomery Ladder Algorithm based ECSM. In case of transient faults, the probability of undetected error is no larger than $1/2^{m-3}$ where m is usually more than 160 bits. Compared with more than 100% hardware overhead of TMR, DMR_PV and PRC methods presented in [10], the hardware overhead and switching activities overhead of LOEDAR are about 27% and 69% respectively. Each additional error detection process contributes less than 1% on average of additional time overhead and switching activities overhead.

REFERENCES

[1] P.L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of Computation*, vol. 48, 1987, pp. 243-264.

[2] Julio Lopez and Ricardo Dahab, "Fast multiplication on elliptic curves over GF(2m) without precomputation," *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems*, 1999, pp. 316-327.

[3] J. Blömer, M. Otto, and J. Seifert, "Sign Change Fault Attacks on Elliptic Curve Cryptosystems," *Fault Diagnosis and Tolerance in Cryptography 2006 (FDTC '06), volume 4236 of Lecture Notes in Computer Science*, Prentice Hall, 2004, pp. 36--52.

[4] M. Ciet and M. Joye, "Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults," *Designs, Codes and Cryptography*, vol. 36, 2005, pp. 33-43.

[5] I. Biehl, B. Meyer, V. Muller, U.K. Wacana, and J.D. Wahidin, "Differential Fault Attacks on Elliptic Curve Cryptosystems," *LECTURE NOTES IN COMPUTER SCIENCE*, Springer-Verlag, 2000, pp. 131-146.

[6] A. Antipa, D. Brown, A. Menezes, R. Struik, and S.A. Vanstone, "Validation of Elliptic Curve Public Keys," *PKC '03: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, 2003, pp. 211-223.

[7] P.-A. Fouque, R. Lercier, D. Real, and F. Valette, "Fault Attack on Elliptic Curve with Montgomery Ladder Implementation," *Fault Diagnosis and Tolerance in Cryptography (FDTC '08)*, IEEE Computer Society, 2008.

[8] SECG, "Standards for Efficient Cryptography 1 (SEC1): Elliptic Curve Cryptography," http://www.secg.org, 2009.

[9] A. Dominguez-Oviedo and M. Anwar Hasan, "Algorithm-level Error Detection for ECSM," *CACR Technical Reports of University of Waterloo, 2009-05*, 2009.

[10] A. Domínguez-Oviedo and M.A. Hasan, "Error Detection and Fault Tolerance in ECSM using Input Randomization," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, 2009, pp. 175-187.

[11] S.N. Cirrus, S. Niranjan, J.F. Frenzel, and M. Ieee, "A Comparison of Fault-Tolerant State Machine Architectures for Space-Borne Electronics," *IEEE Transactions on Reliability*, vol. 45, 1996, pp. 109-113.