Elliptic Curve Cryptoprocessor with Hierarchical Security

ALAAELDIN AMIN,
Computer Engineering Department
King Fahd University of Petroleum & Minerals
Dhahran, 31261, Saudi Arabia
amindin@kfupm.edu.sa

TURKI F. AL-SOMANI
Computer Engineering Department
Um-UL-Qura University
Makka, Saudi Arabia
tfsomani@uqu.edu.sa

Abstract: - This paper describes an elliptic curve scalar multiplication method which is resistant to power analysis attacks. The proposed method confuses both the private key bit values and positions. Even with correct leaked information on the type of operations performed, associating that with a particular key bit value or position is almost impossible. Resistance to side channel attacks is provided at several levels. At the top level, the secret key is segmented into a number of randomly sized segments processed in random order. At the segment level, each segment is encoded randomly using NAF or binary encodings. Further, at the segment level, the inspection direction of segment bits for binary-encoded segments is randomly assigned either in MSB-to-LSB or LSB-to-MSB. Furthermore, at the individual segment bit level, zero bits can randomly trigger a dummy PADD operation. In addition to improved security, this results in an average saving of 50% over the number of dummy PADDs in the Double-and-Add-Always algorithm. Such hierarchical multi-level scheme causes the relation between the private key and possible leaked information to be quite confused resulting in a higher system security with minimal overhead for both speed and area.

Key-Words: - Elliptic Curve Cryptosystems, Side Channel Attacks, Normal Basis, Non Adjacent Form, Public Key Cryptography.

1 Introduction

Elliptic Curve Cryptosystems (ECC) [1], [2] have been gaining increased popularity because of their shorter keys and higher security level. With much shorter key sizes, they represent a viable alternative to earlier public key cryptosystems such as the Rivest-Shamir-Adleman (RSA) [3] and ElGamal [4].

Several software ECC implementations have been reported in recent years [5]. However, software implementations, are both slow and with limited security for private keys compared to hardware implementations. Accordingly, efficient and secure hardware implementations of ECC have been the subject of active research [6], [7], [8], [9].

An elliptic curve over a finite field $GF(q)$ is defined by the set of (x, y) points satisfying the elliptic curve equation together with the "point at infinity" (O) [1]. The (x, y) coordinates of elliptic curve points are elements of $GF(q)$, where $q = p^m$ with p being a prime number. Such set of points forms an abelian group, with point addition (PADD) being the group operation and the "point at infinity", O, the additive identity element of the group. A PADD operation of two elliptic curve points P_1 and P_2 is defined such that the result is a 3rd point R on the curve ($R = P1 + P2$). Adding two elliptic curve points P_1 and P_2 such that $P_1 = P_2 = P$ results in the point doubling operation (PDBL) $R = 2P$.

The Scalar Product (SP) of an elliptic curve point P by a scalar k is a major operation for ECC. Computing the SP (kP) in ECC is analogous to modulo- exponentiation in the multiplicative group of integers. To compute kP a sequence of double-and-add operations is performed based on the binary representation of the ℓ-bit scalar $k = (k_{l-1}, \dots, k_0)$ which is typically a private key. A good survey of reported methods for computing the SP may be found in [10]. Power analysis attacks [11] on ECC can seriously compromise security these devices. Simple Power Analysis (SPA), attempts to gain key information through observing a single trace of the device power dissipation while Differential Power Analysis (DPA) attempts that using multiple such traces [11]. In addition to classical DPA attacks several other recent attacks have been reported, e.g. $Refined\ Power\ Analysis$ (RPA) [14], [20] $Zero\ Power\ Analysis$ (ZPA)[15], $Doubling\ Attack$ [13] and $Address\text{-}Bit\ Differential\ Power\ Analysis$ ($ADPA$) [22]. Analysis of power traces can possibly distinguish PADD from PDBL operations. Coron [17] has shown that the instructions of an SPA resistant Algorithm must be independent of the data being processed, i.e. no data-dependent branch instructions should be used. This may be achieved by

performing both the PADD and PDBL operations every iteration. The result of addition, however, is committed only when $k_i = 1$ and dropped when $k_i = 0$. Such "$Double\text{-}and\text{-}Add\text{-}Always$" approach may be implemented by scanning the key bits either from the most to the least significant bit (MSB-to-LSB) (Algorithm 1) or from the least to the most significant bit (LSB-to-MSB) (Algorithm 2). This scheme, however, is only resistant to SPA attacks while being vulnerable to DPA attacks.

Algorithm 1. The $Double\text{-}and\text{-}Add\text{-}Always$ Algorithm (MSB-to-LSB)

0. ***input*** P, k
1. $Q[0] = P$
2. ***for*** j ***from*** $\ell - 2$ ***to*** 0 ***do***
   2.1. $Q[0] = 2Q[0]$
   2.2. $Q[1] = Q[0] + P$
   2.3. $Q[0] = Q[k_j]$
   end for
3. ***output*** $(Q[0])$

Algorithm 2. The $Double\text{-}and\text{-}Add\text{-}Always$ Algorithm (LSB-to-MSB)

0. ***input*** P, k
1. $Q[0] = P$
2. $Q[1] = O$
3. ***for*** j ***from*** 0 ***to*** $\ell - 1$ ***do***
   3.1. $Q[2] = Q[0] + Q[1]$
   3.2. $Q[0] = 2Q[0]$
   3.3. $Q[1] = Q[1 + k_j]$
   end for
4. ***output*** $(Q[1])$

To protect against DPA attacks, the following 3 countermeasures were suggested [17]:

1. Randomizing the private exponent: If $\#E$ is the number of elliptic curve points, compute the SP $Q = kP$ as follows:
   - An m-bit number d is randomly selected.
   - Compute $k' = k + d\ \#E$.
   - Compute the point $Q = k'P$, where $Q = kP$ since $\#EP{=}O$.
2. Blinding the Point P through adding a secret random point R whose SP $S = kR$ is known, e.g. through precomputation. Then the point $k(R + P)$ is computed then $S = kR$ is subtracted to get $Q = kP$.
3. Randomizing the used Projective Coordinates: Point (X, Y, Z) in projective coordinates actually represents a class of points $(\lambda^c X,\ \lambda^d Y, \lambda Z)$ for different values of $\lambda \neq 0$ in the finite field.

Enhanced version of these countermeasures was proposed in[25], [17] which uses an elliptic curve isomorphism, thereby transposing the computation into another curve through a random morphism. These countermeasures, however, add computational overhead while still being vulnerable to some of the more recent DPA attacks. For example, the first two countermeasures proposed by Coron fail to protect against $doubling\ attack$s. The 3rd countermeasure, however, protects against $doubling\ attack$s as do randomized exponentiation algorithms[19]. A random blinding point R should be updated randomly to protect against $doubling\ attacks$. Such updating should not be regular. Likewise, Smart [19] proposed a defence against $refined\ power\ analysis\ RPA$ attacks. The zero-value point attack (ZPA) can be considered as extension to RPA attack [15]. In RPA attacks, the attacker uses a special point with a coordinate of zero value. In a ZPA attack, an attacker utilizes an auxiliary register which might take a value of zero in the definition field. For protection against ZPA and RPA attacks, the secret key k or the base point P should be randomized. The $BRIP$ countermeasure [21] uses an initial random point R. First $kP + R$ is computed then R is subtracted to obtain kP causing no special point or zero-value register during operations causing it to be resistant against both ZPA and RPA attacks. A hardware-based countermeasure using Randomized Register Renaming (RRR) was proposed [23] against address-bit DPA ($ADPA$) [16]. Several countermeasures against $ADPA$ attacks were proposed [16], but those double the computing time. The randomized addressing (RA) countermeasure, uses an approach which is similar to RRR without requiring special hardware [24]. In RA, randomized register addresses are used for each scalar exponentiation. This causes register addresses to be randomized with subsequent randomized side channel information.

This paper describes an efficient method for elliptic curve scalar multiplication with hierarchical protection scheme against power analysis attacks. The rest of the paper is organized as follows: Section 2 gives the details of the algorithm and the adopted protection schemes at each level. Section 3 analyzes the time, space and security aspects of the algorithm while Section 4 describes the hardware implementation of the proposed algorithm and discusses obtained results with conclusions provided in Section 5.

2 Scalar Multiplication with Hierarchical Protection

The scalar multiplication algorithm proposed here provides hierarchical resistance against power

analysis attacks at several levels. Even with correctly leaked information on the type of operations being performed are leaked, using such information to infer a particular key bit value or position is quite infeasible. The main idea is to confuse both the values and positions of the private key bits.

Several power analysis attacks countermeasures are hierarchically introduced at different levels. This makes it quite hard for attackers to associate leaked information with specific bit value or position.

At the top level, the key is segmented into a random number of randomly sized segments that are processed in a random order. At the segment level, each segment is encoded randomly using NAF [25] or binary. Further, the direction of bit inspection for binary-encoded segments at the segment level is assigned randomly either in MSB-to-LSB or LSB-to-MSB. Furthermore, at the level of individual segment bit, zero bits can randomly trigger a dummy PADD operation. In addition to improved security, this results in an average saving of 50% over the number of dummy PADDs in the $Double-and-Add-Always$ algorithms (Algorithms 1 and 2). It is important to point out that even though each of the above-mentioned measures may be individually vulnerable, the security of our proposed method comes from all these measures combined together.

2.1 The Top-Level

The security measures at the top level are meant to cause confusion in the positions of the private key bits. The key is divided into a number of segments. Both the number of these segments and their sizes are randomized. Furthermore, the order in which the segments are processed is also randomized. Resulting points are then assimilated into the required SP (kP).

One precomputed point is associated with each segment to maintain its significance. A different key segmentation is generated frequently enough to improve resistance to power attacks. Resistance to power analysis attacks improves with larger number of segments (more permutations). This, however, requires more precomputed points and more PADD operations resulting in increased area and delay overhead. Let the number of segments equal u; thus:

$$k = k^{(u-1)} \,||\, k^{(u-2)} || \ldots || k^{(1)} || k^{(0)}$$

Computing the SP is done as follows:

$$\begin{aligned} kP &= \left(k^{(u-1)} || k^{(u-2)} || \ \ldots \ || k^{(1)} || k^{(0)}\right)P \\ &= \left(2^{size(u-1)}\, k^{(u-1)} + 2^{size(u-2)}\, k^{(u-2)} + \right. \\ &\cdots \quad \left. + 2^{size(1)}\, k^{(1)} + k^{(0)}\right)P \\ &= \left(2^{size(u-1)}P\right)k^{(u-1)} + \left(2^{size(u-2)}P\right)k^{(u-2)} + \\ &\quad \ldots + \left(2^{size(1)}P\right)k^{(1)} + (P)k^{(0)} \\ &= P_{u-1}k^{(u-1)} + P_{u-2}k^{(u-2)} + \cdots + P_1 k^{(1)} + P_0 k^{(0)} \end{aligned}$$

where precomputed point P_i is associated with segment $k^{(i)}$, $P_0 = P$, and the size of segment j is $size^{(j)} = \left(\sum_{i=0}^{j-1} size\ of\ segment\ k^{(i)}\right)$

Given that $P_0 = P$, $(u-1)$ P_i points need to be precomputed and stored for u segments. Processing these segments yield curve points that are assimilated to produce the final SP $kP = \sum_{i=0}^{u-1} k^{(i)} P_i$.

Sizes of different segments ($size^{(j)}$) may be either equal or different. Equal sizes yield simpler designs, but different size segments provide higher security. Randomizing the order of segments yields a new pseudo key k' having the same segments as those of k but in a different order. Further, each segment of k' $\left(k'^{(i)}\right)$ must be associated with two other parameters; one is its corresponding precomputed point (P_i), and another is its length in bits $\left(L^{(i)}\right)$. Thus, if the key k is segmented as $k = k^{(u-1)} \,||\, k^{(u-2)} || \ldots || k^{(1)} || k^{(0)}$, the obtained pseudo key after randomization is $k = k'^{(u-1)} || \, k'^{(u-2)} || \ldots || k'^{(1)} || k'^{(0)}$ *where* $k'^{(j)}$ is a triplet defined by:

$$k'^{(j)} = \left(k^{(i)}, P_i, L^{(i)}\right) \qquad (1)$$

where, in general, $i \neq j$.

If the base point P is changed or the segments number or sizes are changed, a new set of precomputed P_i points should be generated.

2.2 The Segment Level

Two countermeasures are used at the segment level. The first is meant to confuse segment bit values. In this case, each segment is randomly encoded to in either NAF [25] or binary. NAF encodings use binary signed digit (BSD) representation with key bit values of 0, 1 or $\bar{1}$ where $\bar{1}$ designates a (−1). With an average of one-third the total number of bits, NAF codes enjoy the least number of nonzero bits of any BSD representation. Every integer has a unique NAF code whose length is, in general, larger than the equivalent binary representation by 1 bit. Algorithm 3 converts a binary encoded key (k) into its equivalent NAF code. The second measure confuses the private key bit positions. If a given segment is binary-encoded, the direction of bit inspection is selected randomly either in MSB-to-LSB or the LSB-to-MSB. Thus, even in case an attacker makes a correct guess that some bit belongs to a particular segment, its exact position within the segment is still confused adding another level of resistance.

Algorithm 3. Efficient NAF encoding algorithm

Inputs: Unsigned ℓ -bit Binary Code *k*.
Output: NAF(k).
1. N = 0 ; $k'_\ell = 0$.
2. ***for*** $j = 0$ ***to*** $\ell - 1$ ***do***
2.1. $k'_j = (N + k_j) \bmod 2$
2.2. **if** $k'_j = 1$ **and** $k'_{j+1} = 1$ **then**
2.2.1. $k'_j = 1$
2.2.2. N = 1
2.3. **else**
2.3.1. N = N . k_j
3. **if** N = 1 **then** $k'_j = 1$
4. ***Output*** (*k'*)

2.3 The Bit Level

At this level, more confusion is added to the private key bit values. In the $Double-and-Add-Always$ algorithms (Algorithms 1 and 2), PDBL and PADD operations are performed every iteration irrespective of the value of the key bit being processed (k_i). For these algorithms, if $k_i = 1$, the results of PDBL and PADD are committed, otherwise PADD result is ignored and only the PDBL result is committed.

In spite of being simple to implement, the $Double-and-Add-Always$ algorithms make the SP resistant only to SPA but not to DPA attacks. Further, the extra dummy PADDs cause this approach to suffer from excessive delay overhead.

At the bit level, we have adopted another resistance measure through randomly performing dummy PADD if $k_i = 0$. Thus, based on the value of a random Boolean parameter *r*, a dummy PADD may be performed if $k_i = 0$ and $r = 1$.

Algorithm 4 shows the SP procedure SP_NAF for NAF-encoded keys (or segments) which performs random dummy PADDs for zero key bits. In this algorithm, the NAF bits are generated but rather the corresponding point operations, i.e. point addition or subtraction, are performed. For example, step 2.2.1 performs a point subtraction without need to generate the actual value of corresponding key bit value ($\bar{1}$). If the NAF code has one more bit than the equivalent binary representation, the PADD operation at step 4.1 is performed. For binary-encoded keys (or segments), algorithms 5 and 6 show the the MSB-to-LSB and LSB-to-MSB SP procedures (SP_ML and SP_LM respectively) which perform random dummy PADDs for zero key bits. In all of the three algorithms, PDBL is always performed while PADDs are performed depending on the values of k_i and the random variable r.

Algorithm 4. NAF SP Procedure with Random Dummy Point Additions.

Procedure: $SP\_NAF(k, P, \ell)$
Inputs: *k*: Unsigned Binary Code, *P*: elliptic curve point, and ℓ: Size of k in bits.
Output: Elliptic Curve Point which is equal to the SP kP.
1. N = 0 ; $Q[0] = P$; $Q[1] = O$.
2. ***for*** $j = 0$ ***to*** $\ell - 1$ ***do***
2.1. $k'j = (N + k_j) \bmod 2$
2.2. **if** $k'_j = 1$ **and** $k_{j+1} = 1$ **then**
2.2.1. $Q[2] = Q[1] - Q[0]$
2.2.2. $N = 1$
2.3. **else**
2.3.1. $N = N.k_j$
2.3.2. **if** ($k'_j = 1$ **or** $r = 1$) **then**
$Q[2] = Q[1] + Q[0]$
2.4. $Q[0] = 2Q[0]$
2.5. $Q[1] = Q[1 + k'_j]$
3. **if** $N = 1$ **then** $Q[1] = Q[1] + Q[0]$
4. ***Output*** ($Q[1]$)

Example

Fig. 1 illustrates the flow of events in the proposed scalar multiplication procedure for a 16-bit key example *k*=1001_1000_1011_0111. First, the key is randomly segmented into a number of segments. Let the number of segments be four, thus:

$k = k^{(3)} \,||\, k^{(2)} \,||\, k^{(1)} || k^{(0)}$ (2)

where,

$k^{(0)} = 11$, $L^{(0)} = 2$, $P_0 = 2^0 P$
$k^{(1)} = 01101$, $L^{(1)} = 5$, $P_1 = 2^2 P$
$k^{(2)} = 001$, $L^{(2)} = 3$, $P_2 = 2^7 P$
$k^{(3)} = 100110$, $L^{(3)} = 6$, $P_3 = 2^{10} P$

Next, the segments are randomly re-ordered forming a new randomized pseudo key k' as follows:

$k' = k'^{(3)} \,||\, k'^{(2)} \,||\, k'^{(1)} || k'^{(0)}$, *where,*
$k'^{(0)} = \left(k^{(3)}, P_3, L^{(3)}\right)$
$k'^{(1)} = \left(k^{(1)}, P_1, L^{(1)}\right)$
$k'^{(2)} = \left(k^{(2)}, P_2, L^{(2)}\right)$
$k'^{(3)} = \left(k^{(2)}, P_2, L^{(2)}\right)$

The following step randomly selects either a NAF or a binary encoding for each segment. In this example, we assume that only $k'^{(0)}$ (originally $k'^{(3)}$) is NAF encoded while the other segments are binary-encoded. The binary-encoded segments are then randomly assigned a direction of inspection (MSB-to-LSB or LSB-to-MSB). Here, we assume that $k'^{(3)}$ and $k'^{(2)}$ are assigned the MSB-to-LSB direction while $k'^{(1)}$ is assigned the LSB-to-MSB inspection

Algorithm 5. Binary *MSB*-to-*LSB* SP with Random Dummy Point Additions.

Procedure: $SP\_ML(k, P, \ell)$
Inputs: *k*: Unsigned Binary Code, *P*: elliptic curve point, and ℓ: Size of k in bits.
Output: Elliptic Curve Point which equals the SP *k*P..
1. Q[0] = P . $k_{(\ell-1)}$ /* $k_{(\ell-1)}$ = leftmost bit of k */
2. ***for*** i ***from*** $\ell - 2$ ***to*** 0 ***do***
   2.1. $Q[0] = 2Q[0]$
   2.2. **if** ($k_i = 1$ **or** $r = 1$) **then** $Q[1] = Q[0] + P$
   2.3. $Q[0] = Q[k_i]$
   end for
3. ***Output*** ($Q[0]$)

direction. Then the proper algorithm is invoked to compute the partial SP for each segment. Thus, Algorithm 5 is invoked for segments $k'^{(3)}$ and $k'^{(2)}$, Algorithm 6 is invoked for segment $k'^{(1)}$, and Algorithm 4 is invoked for segment $k'^{(0)}$. During the execution of these algorithms, when a zero bit is encountered in the bitwise inspection of segments, a dummy PADD is performed if the binary random variable (*r*) equals 1 as clarified in step 4 of Fig. 1.

Algorithm 6. Binary *LSB*-to-*MSB* SP with Random Dummy Point Additions.

Procedure: $SP\_LM(k, P, \ell)$
Inputs: *k*: Unsigned Binary Code, *P*: elliptic curve point, and ℓ: Size of k in bits.
Output: Elliptic Curve Point which equals the SP *k*P.
1. $Q[0] = P$
2. $Q[1] = O$
3. ***for*** i ***from*** 0 ***to*** $\ell - 1$ **do**
   3.1. **if** ($k_i = 1$ **or** $r = 1$) **then** $Q[2] = Q[1] + Q[0]$
   3.2. $Q[0] = 2Q[0]$
   3.3. $Q[1] = Q[k_i + 1]$
   end for
4. ***Output*** ($Q[1]$)

2.4 Scalar Multiplication

The pseudo-code of the proposed SP algorithm with hierarchical security is shown in Algorithm 7. The key is divided into *u* segments with associated (u−1) precomputed points. Precomputed points are calculated by repeated PDBL operations as shown in Step 2. Each segment $k^{(i)}$ is associated with a precomputed point P_i (step 3). Up to step 3 may be performed off line and the outcome results can be re-used more than once for the same *k* and *P* values.

Algorithm 7. Scalar Multiplication Algorithm with Multi-Level Security Measures.

Inputs *P*: Base Point, *k*: Private key, *u*: # of segments, $L^{(u-1)}$, ..., $L^{(1)}$, $L^{(0)}$: Sizes of all segments
Output: *kP*
1. Initialization: Q = P, P_0 = P.
2. /* *Calculating the Pre-computed points*:*/
   2.1 ***for*** $i = 1$ ***to*** $u - 1$ ***do***
      2.1.1 ***for*** $j = 0$ ***to*** $L^{(i-1)} - 1$ ***do***
         2.1.1.1. $Q = 2Q$
      2.1.2. ***end for***
      2.1.3. $P_i = Q$
   2.2. ***end for***
3. Associate each segment $k^{(i)}$ with a pre-computed point P_i and size $L^{(i)}$.
4. /* Randomization Process */
• the order of segments is randomized resulting in a pseudo key:
$k' = k'^{(u-1)} || k'^{(u-2)} || \ldots || k'^{(1)} || k'^{(0)}$
with $k'^{(j)} = \left(k^{(i)}, P_i, L^{(i)}\right)$ *where*, in general, $j \neq i$ and $i, j = 0, 1, \ldots, u - 1$.
• type of encoding for each segment is selected either as NAF or binary at random.
• binary-coded segments bits randomly inspected either in *MSB*-to-*LSB* or *LSB*-to-*MSB*.
• Initialize $R = O$,
5. ***for*** $i = 0$ ***to*** $u - 1$ ***do***
   5.1 ***if*** (Binary-encoded Segment) ***then***
      5.1.1. ***if*** (*MSB*-to-*LSB*) ***then***
         $Q = SP\_ML(k'^{(i)})$ /* Alg. 5 */
      5.1.2. **else** Q = $SP\_LM(k'^{(i)})$ /* Alg. 6 */
   5.2. **else** Q = $SP\_NAF(k'^{(i)})$ /* Alg. 4 */
   5.3. $R = R + Q$
   end for
6. ***Output***(R)

The actual scalar multiplication process starts at step 4 where segments are randomly re-ordered, the encoding type of each segment randomly decided, and the binary-encoded segments direction of inspection is randomly picked. This is followed by the main scalar multiplication loop at step 5. Steps 5.1.1 and 5.1.2 process binary-coded segments if the direction of inspection is *MSB*-to-*LSB* or *LSB*-to-*MSB* using algorithms 5 and 6 respectively. For segments designated for *NAF* encoding, Algorithm 4 is invoked at step 5.2. A segment is processed in the same manner as the key is. To compute the final SP, the points resulting from individual processing of segments are accumulated (step 5.3) in one point *R*, for a total of (u−1) additional PADDs.

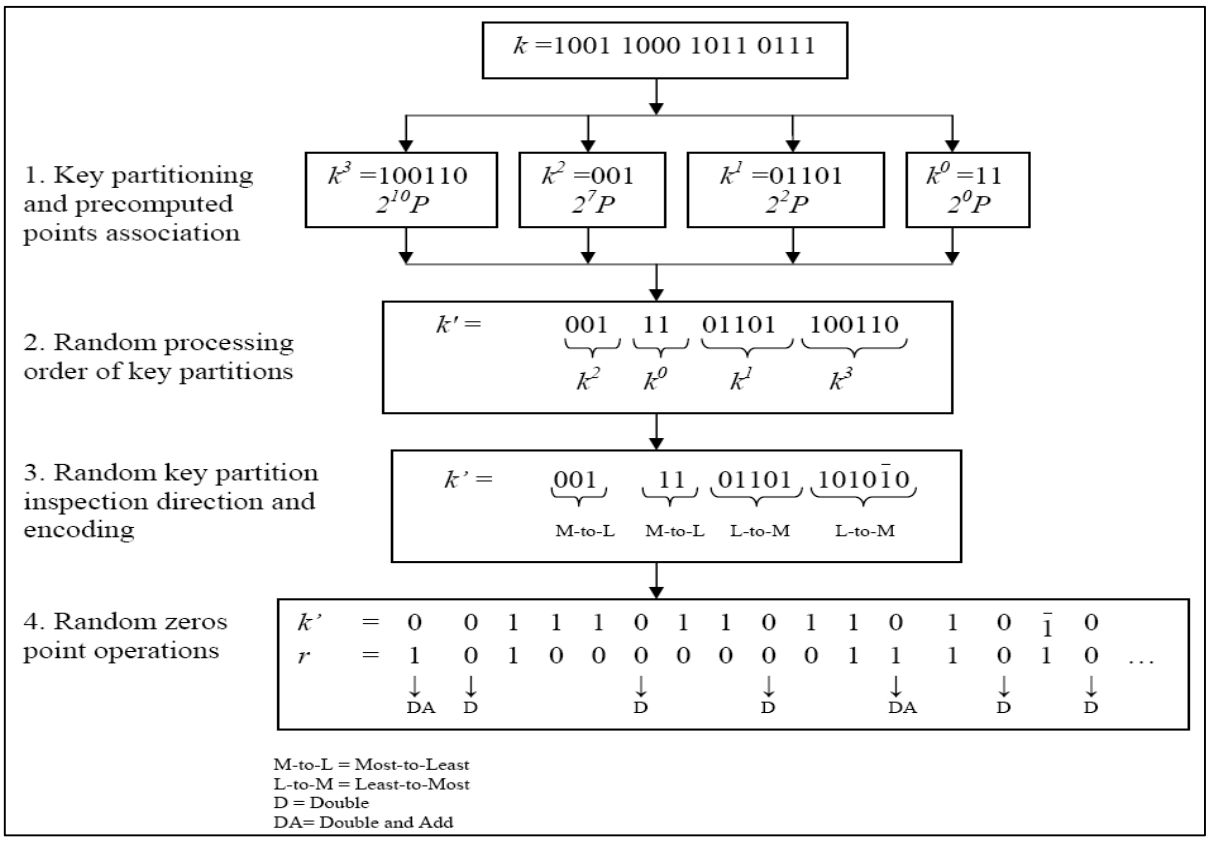

Fig. 1: Scalar Multiplication with Hierarchical Security Flow of Events.

3 Algorithm Analysis

Even if a prospective attacker managed to distinguish between the PADD and the PDBL operations, the proposed hierarchical counter-measures makes it extremely difficult for attackers, given such information, to infer neither the value nor the position of key bits. The hierarchical resistance measures confuse prospective attackers as to the exact key *bit positions* due to:

1. The random order of segment processing,
2. The randomly assigned number, and generally unequal sizes of segments,
3. The randomly assigned direction of inspection for binary-encoded segments, and
4. The segments sizes that may appear larger by one bit when these segments are NAF-encoded.

In addition, prospective attackers cannot ascertain definite values for *key*. A key bit value that appears to an attacker as 1, may actually be one of four possible values; (i) a value of 1 in a NAF-encoded segment, or (ii) a value of $\bar{1}$ in a NAF-encoded segment, or (iii) a true binary 0 value but with a dummy PADD operation randomly performed, or (iv) a true binary 1 value. Likewise, a bit that appears to have a 0 value, may be (i) a NAF-encoded 0, or (ii) a true binary 0 value.

The use of multiple precomputed points (P_i) for segments effectively blinds the original base point P particularly when the number of segments or their sizes are changed frequently enough.

With different encodings for different segments and different random dummy point adds performed with zero key bit values coupled with the different processing order and direction of these segments, multiple runs of the proposed scalar multiplication algorithm will result in different power traces uncorrelated with other runs. With different uncorrelated power traces for different runs, the possibility of mounting successful DPA attacks seems quite unlikely.

Protecting against ZPA and RPA requires that either the secret key k or the base point P be randomized. For the proposed SP algorithm, both effects are present since the base point P appears to be blinded and the private scalar k appears to be randomized as detailed earlier. Accordingly, the proposed hierarchical protection algorithm is secure against RPA and ZPA. Furthermore, resistance to the $doubling\ attack$ is achieved since $doubling$

$attacks$ apply to the double-and-add algorithm only in the MSB-to-LSB direction[13]. Our proposed scalar multiplier is designed to perform a randomized mix of either the MSB-to-LSB or the LSB-to-MSB of the double-and-add algorithm at the segment level. Moreover, randomization techniques used at each level of this hierarchical protection scheme cause any correlation between register addresses and the secret key very difficult to establish. Thus, the proposed algorithm is resistant to $ADPA$ attacks.

With the random encodings, $\frac{1}{3}$ of the bits of NAF-encoded segments (on the average) have non-zero values as compared to half the bits for binary-encoded segments. For an m-bit private scalar multiplier k, on the average, half the bits are binary encoded while the other half is NAF-encoded. Thus, on the average, binary encoded bits require $\frac{m}{4}$ PADDs, while NAF encoded bits require only $\frac{m}{6}$ PADDs. Accordingly, an average of $\frac{5}{12}m$ PADDs are performed for the segments $1's$ and the $\bar{1}'s$. At the lowest (bit) level, a 0 bit value may trigger a dummy PADD. Since segment encodings may be either NAF or binary, $\frac{7}{24}m$ dummy PADD (on the average) are performed. Thus, the average total number of PADDs is $\frac{7}{24}m \cong 0.7m$. The added random dummy PADDs in the dummy point add algorithms (Algorithms 4, 5 and 6) increase the degree of confusion, however, they also do increase the number of PADDs and accordingly the total delay overhead. In spite of the possibly significant added delay overhead, this approach is quite attractive compared to the $Double-and-Add-Always$ algorithms (Algorithms 1 and 2).

Assimilating the final SP result incurs additional (u−1) PADDs. Accordingly, an average of ($0.7m + u - 1$) PADDs are performed by our proposed algorithm.

For an m-bit private scalar multiplier k, m PDBL operations are performed. Neglecting pre-computations, the proposed scalar multiplier with hierarchical security, thus, requires m PDBL operations and an average of ($0.7m$+u−1) PADD operations. With $u \leq 0.1m$ for large values of m, the performance of the proposed algorithm is better than the $Double-and-Add-Always$ algorithm, the Random Initial Point method (BRIP) [21], the Modular Scalar Randomization (MSR) [29] which all require m PDBL operations and m PADD operations. The performance of the proposed algorithm is also better than the three methods presented in [12] .

If the base point P or the segment sizes / number are changed, a new set of precomputed points P_i must be generated. Calculating the precomputed points P_i requires $\left(m - L^{(u-1)}\right)$ PDBL operations. Assuming an average length for the (u-1)th segment, the average number of PDBL operations needed to calculate the precomputed points becomes $\left(m - \frac{m}{u}\right)$. While a larger number of segments u provides higher security, it also results in additional area and delay overhead. Increasing u will increase the number of precomputed points to be stored (u−1) and would also increase the delay overhead since additional $(u-1)$ PADDs are needed to assimilate the partial results into the final SP (kP). In addition, an increased number of segments increases the average number of PDBL operations required to calculate the precomputed points.

4 Implementation and Results

To evaluate the area and delay overhead caused by the hierarchical security measures adopted by our proposed SP algorithm, two ECC processors have been designed, modelled in VHDL, and synthesized onto a Xilinx xc2v8000 FPGA chip. The two processors differ only in the SP algorithm used. One of the processors is non-secure (ECC_{NS}) using a simple double and add SP algorithm, while the other is a strongly secure processor (ECC_{SS}) implementing the SP algorithm with our proposed hierarchical security measures (Algorithm 7). The VHDL models are parameterized to allow investigating different architectural parameters of the synthesized cryptoprocessors. The main definable parameters include the elliptic curve equation coefficients a and b, the underlying $GF(2^m)$ field parameter m, the base point P, the secret key k, and the segment sizes and number for the strongly secure ECC_{SS} cryptoprocessor.

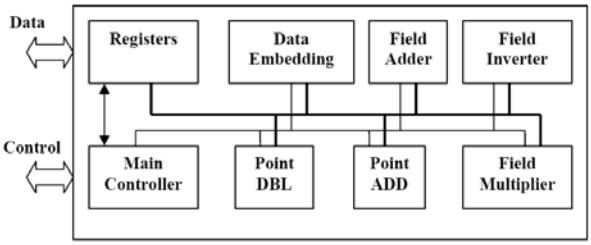

Fig. 2: The proposed architecture

The elliptic curve Diffie-Hellman protocol was selected for the encryption/decryption process. Both processors use optimal normal basis (ONB) over $GF(2^m)$. The basic units (Figure 2) of these processors include: (i) main controller, (ii) data embedding unit, (iii) the PADD and PDBL units, and (iv) the field arithmetic units (adder, multiplier and

inverter). To embed the plaintext into a random curve point on the in the encryption process, a data-embedding unit is used.

PADD and PDBL operations use the projective coordinate system of Lopez-Dahab [26] where $(x, y) = (X/Z, Y/Z^2)$. In the Lopez-Dahab coordinate system, PADD and PDBL operations require 14 and 5 field multiplications respectively. Field squaring for optimal normal basis representation is a simple cyclic shift operation while field addition is simply a bitwise XOR operations requiring a single clock cycle for either operation. Field multiplication, however, is more complex and is crucial for efficient computations. We adopted the Sunar–Koc multiplier [27] because of its low space and time complexities. Because of the limited FPGA resources, a sequential implementation of the Sunar–Koc multiplier was used to save on the limited resources of FPGAs.

For encryption/decryption, a single field inversion is needed as we are using projective coordinates. We have adopted the Itoh and Tsujii inverter [28] since it requires only $O(log_2(m))$ multiplications. For fair comparison, the same field operation algorithms, e.g. multiplication and inversion were used for both cryptoprocessors (ECC_{NS} and ECC_{SS}). The only architectural difference is to warrant the implementation of the two different SP algorithms in addition to the extra storage requirement of the ECC_{SS} to hold the necessary precomputed points. Thus, performance difference and area overhead are due to the difference in the adopted SP algorithms independent of field operations or the adopted coordinate system.

The two cryptoprocessors have been synthesized on the same Xilinx FPGA (xc2v8000) with values of $m = 14, 30, 65, 90$ and 173 bits. Due to the limited FPGA resources, and the inability to use much higher values of m, some impractically small values of m, e.g. 14 and 30, have been used to provide more points on the area and delay curves that may establish trends for these parameters. The strongly secure processor ECC_{SS} has been also investigated for a number of segments $u = 2, 3,$ *and* 4. The used number of FPGA slices is used as an estimate of the area the processor occupies on the designated FPGA. Since more precomputed points need to be stored, the ECC_{SS} is expected to have larger area for larger number of segments. The delay and area overhead figures of the ECC_{SS} processor are normalized with respect to ECC_{NS} reference cryptoprocessor. Thus, the delay and area overhead of the ECC_{SS} processor are given by:

$$Delay\ Overhead = \frac{ECC_{SS}\ Delay}{ECC_{NS}\ Delay}, \text{ and}$$

$$Area\ Overhead = \frac{ECC_{SS}\ Area}{ECC_{NS}\ Area},$$

Synthesis results are shown in Table 1 and the normalized area-delay overhead values of the ECC_{SS} for various values of m and u are plotted in Fig. 3. The figure shows that, for the same number of segments u, an increased field size m reduces both the area and delay overheads since the number of additional PADD operations and precomputed points storage become less significant. For a given field size, the number of segments can be chosen within a certain range of acceptable area and delay costs. Increasing the number of segments u result in higher resistance to power analysis attacks at the expense of more delay and area overheads.

5 Conclusion

The described elliptic curve SP method uses a combination of hierarchical security measures at different levels to improve resistance against power analysis attacks. At the top level, the secret key is segmented into a random number of segments of random sizes to confuse key bit values and positions. At the lower segment level, segments are processed in a randomized order. In addition, segments are randomly encoded either in NAF or in binary. In addition, at the segment level, bits in binary-encoded segments are inspected randomly in either the MSB-to-LSB or LSB-to-MSB direction. At the lowest bit level, each 0 bit value, may randomly trigger a dummy PADD operation. The hierarchical protection scheme adopted by this algorithm confuses both the private key bit values and positions. Even if correct information regarding the type of operations being performed are leaked, associating that with particular key bit values or positions will be very difficult. For u segments, additional $(u - 1)$ PADDs are needed to compute the SP. Likewise; additional $(u - 1)$ precomputed points need to be stored. Larger number of segments yields higher level of security but would incur area and delay overheads. With m PDBL operations and an average of $0.7m + u - 1$ PDBL operations, proper choice of u can be made through simulations to combine both security and low overhead. A VHDL model of the proposed secure elliptic curve cryptoprocessor employing the new scalar multiplication method has been developed and synthesized onto a Xilinx xc2v8000 FPGA. Using this model, the overhead area and delay values resulting from introducing the proposed hierarchical security measures for several values of the number of segments have been estimated. Adequate security

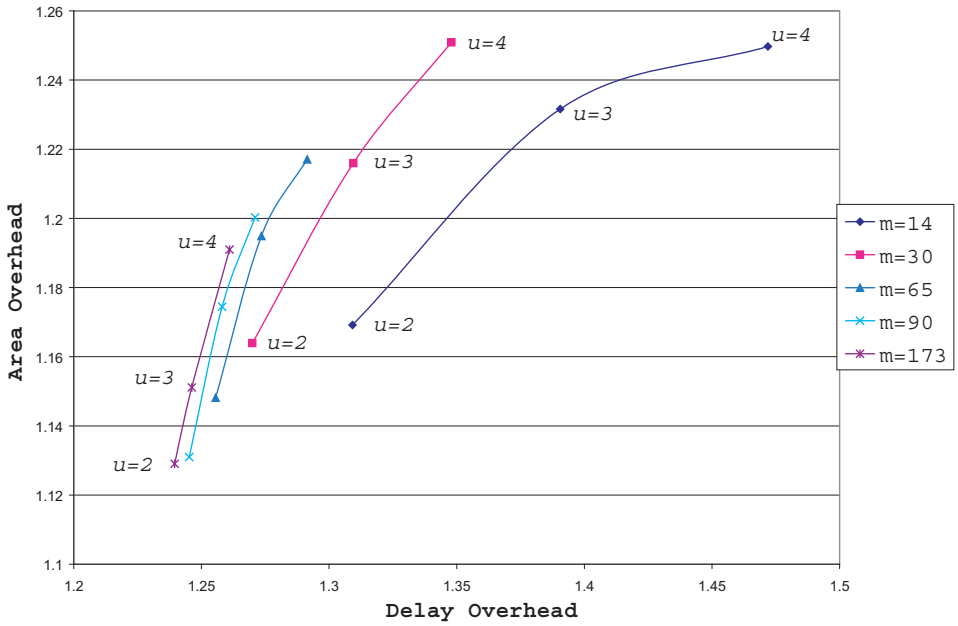

Fig. 3: The ECC_{SS} Cryptoprocessor Delay and Area Overheads.

may still be attained even if partial utilization of the proposed hierarchical protection techniques is used (e.g. eliminating the randomization in either the segment direction or encoding).

Table 1: ECC_{SS} Cryptoprocessor Synthesis Results.

m	u	Clock (MHz)	Delay (µsec)	Delay OH	Area (Slices)	Area OH
14	2	93.954	40.77	1.31	1873	1.169
14	3	93.954	43.30	1.39	1973	1.232
14	4	93.954	45.83	1.47	2002	1.249
30	2	74.172	205.79	1.27	3934	1.164
30	3	74.108	212.20	1.31	4110	1.216
30	4	74.108	218.44	1.35	4228	1.251
65	2	64.295	1037.29	1.25	9696	1.148
65	3	64.295	1052.10	1.27	10091	1.195
65	4	64.295	1066.91	1.29	10278	1.217
90	2	60.055	2091.88	1.24	13496	1.128
90	3	60.055	2113.56	1.26	14015	1.174
90	4	60.055	2135.24	1.27	14323	1.200
173	2	53.454	8492.52	1.24	31053	1.129
173	3	53.454	8538.62	1.25	31660	1.151
173	4	53.114	8639.67	1.26	32757	1.191

Acknowledgement

The authors would like to acknowledge King Fahd University of Petroleum & Minerals and Um-UL-Qura University for support.

References:

[1] N. Koblitz. Elliptic curve cryptosystems. Mathematics of Computation, vol. 48, pp. 203-209, 1987.

[2] V. Miller, "Use of elliptic curves in cryptography," Advances in Cryptology, CRYPTO 85, pp.417-426, 1985.

[3] R. Rivest, A. Shamir, L. Adleman. A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM, Vol. 21, No.2, pp. 120-126, 1978.

[4] T. El Gamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. Advances in Cryptology: Proceedings of CRYPTO 84, Springer Verlag, pp. 10-18, 1985.

[5] S. Khajuria and H. Tange, "Implementation of diffie-Hellman key exchange on wireless sensor using elliptic curve cryptography", in Proc. 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace and Electronic Systems Technology (Wireless VITAE'09), pp. 772-776, May 2009.

[6] Kimmo Jarvinen, "Optimized FPGA-based elliptic curve cryptography processor for high-speed applications," Integration, the VLSI Journal, Volume 44, Issue 4, pp. 270-279, 2011.

[7] K. Jarvinen and J. Skytta, "On Parallelization of High-Speed Processors for Elliptic Curve Cryptography," IEEE Trans. on VLSI, Vol. 16, Issue 9, pp. 1162-1175, 2008.

[8] B. Ansari and M. A. Hasan, "High-Performance Architecture of Elliptic Curve Scalar Multiplication," IEEE Trans. on Computers, Vol. 57, No. 11, pp. 1443-1453, 2008.

[9] G. Meurice de Dormale and J.-J. Quisquater, "High-speed hardware implementations of Elliptic Curve Cryptography: A survey," Journal of Systems Architecture: the EUROMICRO Journal, Volume 53, Issue 2-3, pp. 72-84, February, 2007.

[10] D. Gordon, "A Survey of Fast Exponentiation Methods", Journal of Algorithms, 1998, pp. 129-146.

[11] Kocher, J. Jaffe and B. Jun "Differential Power Analysis", Advances in Cryptology: Proceedings of CRYPTO '99, LNCS 1666, Springer-Verlag, (1999) pp. 388-397.

[12] N. Zhang, Z. Chen and G. Xiao. Efficient Elliptic Curve Scalar Multiplication Algorithms Resistant to Power Analysis, Information Sciences, 177, pp. 2119-2129, 2007.

[13] P. A. Fouque and F. Valette, "The doubling attack - why upwards is better than downwards", In Cryptographic Hardware and Embedded Systems - CHES '03, LNCS 2779, pp. 269-280, Springer-Verlag, 2003.

[14] L. Goubin, "A refined power-analysis attack on elliptic curve cryptosystems", In Public Key Cryptography - PKC'03, LNCS 2567, pp. 199-210, Springer-Verlag, 2003.

[15] T. Akishita and T. Takagi, "Zero-value point attacks on elliptic curve cryptosystem," In Information Security Conf - ISC '03, LNCS 2851, pp. 218-233, Springer-Verlag, 2003.

[16] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka "Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA", Cryptographic Hardware and Embedded Systems: Proceedings of CHES '2002, LNCS 2523, Springer-Verlag, (2002) pp. 129-143.

[17] J. Coron, "Resistance against differential power a nalysis for elliptic curve cryp- tosystems," In Cryptographic Hardware and Embedded Systems - CHES '99, LNCS 1717, pp.292-302, Springer-Verlag, 1999.

[18] M. Joye and C. Tymen, "Protections against Differential Analysis for Elliptic Curve Cryptography," In Cryptographic Hardware and Embedded Systems - CHES '01, LNCS 2162, pp.377-390, Springer-Verlag, 2001.

[19] J. C. Ha and S. J. Moon, "Randomized signed-scalar multiplication of ECC to resist power attacks," In Cryptographic Hardware and Embedded Systems - CHES '02, LNCS 2523, pp. 551-563, Springer-Verlag, 2002.

[20] N. P. Smart, "An analysis Goubin's refined power analysis attack," Proc. of Cryptographic Hardware and Embedded Systems - CHES '03, LNCS 2779, pp. 281-290, Springer-Verlag, 2003.

[21] H. Mamiya, A. Miyaji, and H. Morimoto, "Efficient countermeasure against RPA, DPA, and SPA," In Cryptographic Hardware and Embedded Systems - CHES '04, LNCS 3156, pp. 343-356, Springer-Verlag, 2004.

[22] T. Messerges, E. Dabbish, and R. Sloan, "Investigations of Power Analysis Attacks on Smartcards", preprint, USENIX Workshop on Smartcard Technology, 1999.

[23] D. May, H.L. Muller, and N.P. Smart, "Random Register Renaming to Foil PA", CHES 2001, LNCS 2162, pp. 28-38, Springer-Verlag, 2001.

[24] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka "A Practical Countermeasure against Address-Bit Differential Power Analysis", Cryptographic Hardware and Embedded Systems: Proceedings of CHES 2003, LNCS 2779, Springer-Verlag, (2003) pp. 382-396.

[25] Marc Joye and Christophe Tymen, "Compact Encoding of Non-Adjacent Forms with Applications to Elliptic Curve Cryptography", Public Key Cryptography, vol. 1992 of Lecture Notes, in Computer Science, pp. 353-364, Springer-Verlag, 2001.

[26] Lopez, J. and Dahab, R., Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$. SAC'98, LNCS 1556, pp. 201-212, Springer-Verlag, 1998.

[27] Sunar, B. And Koc, C. K., An efficient optimal normal basis Type II multiplier. IEEE Trans. Comput., Vol. 50, No. 1, pp. 83–88, 2001.

[28] T. Itoh and S. Tsujii, A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. Info. Comput., Vol. 78, No.3, pp. 171-177, 1988.

[29] M. Ciet, M. Joye, (Virtually) Free randomization technique for elliptic curve cryptography, in: Proc. of ICICS-2003, LNCS, vol. 2836, Springer-Verlag, Berlin, pp. 348-359, 2003.