A Scalable and High Performance Elliptic Curve Processor with Resistance to Timing Attacks

Alireza Hodjat[1], David D. Hwang[1], Ingrid Verbauwhede[1,2]
[1] *University of California, Los Angeles*
[2] *Katholieke Universiteit Leuven*
{ahodjat, dhwang, ingrid} @ ee.ucla.edu

Abstract

This paper presents a high performance and scalable elliptic curve processor which is designed to be resistant against timing attacks. The point multiplication algorithm (double-add-subtract) is modified so that the processor performs the same operations for every 3 bits of the scalar k independent of the bit pattern of the 3 bits. Therefore, it is not possible to extract the key pattern using a timing attack. The data flow graph of the modified algorithm is derived and the underlying Galois Field operators are scheduled so that the point multiplication delay is minimized. The architecture of this processor is based on the Galois Field of $GF(2^n)$ and the bit-serial field multiplier and squarer are designed. The processor is configurable for any value of n and the delay of point multiplication is $[18(n+3) + (n+3)/2 + 1]\times(n/3)$ clock cycles. For the case of $GF(2^{163})$ the point multiplication delay is 165888 clock cycles.

Keywords

Elliptic Curve Cryptography, side-channel attacks, Galois fields, hardware architecture, security.

1. Introduction

Elliptic curve cryptography (ECC) is a promising form of public key cryptography for next-generation embedded applications. Because the elliptic key discrete logarithm problem has no known solution that can be computed in sub-exponential time, an ECC system can provide security equivalent to an RSA system while using much smaller parameters (i.e. bit size). Due to these smaller parameters, ECC systems are particularly attractive for deployment in deeply embedded systems with limited resources.

However, it is critical to note that security in an embedded context requires a cryptosystem to be both efficient in resources as well as keenly aware of side-channel attacks. A cryptosystem which is not resource-efficient is impractical for use in a resource-constrained device. Likewise, a system that is not side-channel aware may result in an easily compromised device.

In past years, there have been a number of papers on ECC implementations designed for resistance to side-channel attacks (SCAs). Side-channel attacks on ECC systems are often focused on exploiting the difference in power signatures between the point double and point addition operations. In using a simple double-add scalar multiplication algorithm, for instance, the private key can readily be determined by analyzing in time the power signatures of additions and doubles. Coron [2] demonstrates that differential power analysis can be applied to ECC systems, and shows initial techniques to thwart DPA. Reference [3] provides an algorithm based on Montgomery's method which requires a point double and point addition to occur at each scalar multiplication step, regardless of the key bit. In [4], the authors first analyze the work of [2] and [3] and then demonstrate that a hybrid technique of both is actually the most secure. Oswald and Aigner [5] have shown that adding randomization to the scalar multiplication algorithm to blind the input parameters of the multiplication is a means to provide SCA resistance. The work of [6] provides two techniques to thwart attacks: the first causes point doublings and additions to be indistinguishable; the second performs non-deterministic point exponentiation. An SCA-resistant method using encoding to ensure point doublings and additions occur in a uniform pattern is shown in [7].

This paper presents a scalable ECC hardware implementation that provides resistance to certain types of timing-based side-channel attacks. A modified point multiplication algorithm is presented which is used to conceal the difference between the point addition/subtraction and point double operations. The implementation in this paper is different from prior art because we will focus on hiding the information of the private key in the highest level of the point multiplication algorithm by following a similar dataflow graph (same operations in the same number of steps) for every three bits of the secret-key. Moreover, through our proposed algorithm schedules, the datapath operators are running for every step of the algorithm regardless of the bit-pattern of the private key.

The rest of this paper is organized as follows. In section two the elliptic curve cryptosystem is introduced. Section three presents the modified point multiplication algorithm that is used to gain resistance against timing attacks. Moreover, the performance optimizations and the underlying operator schedules to implement the modified point multiplication algorithm are presented. Section four presents the hardware architecture of the ECC processor that is implemented using the result of section three. Section five provides the performance results and conclusion is in section six.

2. Elliptic Curve Cryptosystem

IEEE public-key standard specification (IEEE P1363) [8] defines the Elliptic Curve Cryptography algorithm. The main operation in a typical elliptic curve cryptosystem is called the point-multiplication which refers to calculating $k.P$ where k is an integer and P is a point on the specific elliptic curve. The theory of ECC is based on the mathematical mapping of an elliptic curve on a Galois Field. The elements of the Galois Field that satisfy the elliptic curve equation form a group with a specific addition operation. With this definition, $k.P$ is equivalent to adding P to itself k times by the group operation. Calculating $2.P$ is referred to as double operation and the inverse of the addition operation is called subtraction. All three operations, doubling, addition, and subtraction are used in the point multiplication algorithm.

Figure 1 shows the point multiplication algorithm [8] that is based on the signed digit representation of integer k and is considered to be a faster point multiplication algorithm compared to the algorithm based on the regular binary representation [9]. This algorithm uses the elliptic curve group operations (double, addition, and subtract) based on the underlying Galois Field. The details of these operations are presented in the next section. The projective coordinates *(X, Y, Z)* are used for the representation of the points on the elliptic curve in order to avoid the inversion operation in the underlying Galois Field [8].

3. Modified point multiplication algorithm

3.1. Resistance against timing attack

As shown in Figure 1, depending on the bit pattern of integer k, a combination of group operations are used. A timing attack is possible because of the data dependent if-conditions, shown in steps 3.2. and 3.3. [2]. This security hole makes it possible to extract the bit pattern of the scalar k (key) using timing attack. Clearly the time that it takes to perform a single doubling or a doubling followed by an addition/subtraction is different and therefore, at each step it is easy to see if the current bit of k is 0 or 1.

In order to hide the key pattern information, we propose to consider more than one bit of k (and h) at a time. Figure 2 shows all the possibilities of double/addition/subtraction for all the different combinations of the three bits of k and h. By assuming that the result of the previous calculation is S, then the new value is calculated using the value of S and the initial point P.

The interesting outcome is that all 27 different combinations can be calculated by doing three doubles in a row ($8S$) and then applying one addition or subtraction with a value mP (*where* $m = 0, 1, 2, ..7$). Since P is a known point on the elliptic curve before the point multiplication algorithm starts, all the values mP are known values and can be pre-calculated and stored in the memory. This means that independent of

```
Input: An integer $k$ and an elliptic curve point $P = (X, Y, Z)$.
Output: The elliptic curve point $S = k.P = (X^*, Y^*, Z^*)$.
1. Set $S = P$
2. Let $k_l k_{l-1} ... k_1 k_0$ and $h_l h_{l-1} ... h_1 h_0$ be the binary
   representations of $k$ and $h=3k$, respectively.
3. For $i$ from $l - 1$ downto $1$ do
    3.1 Set S = 2S
        $(X^*, Y^*, Z^*)$ = Double $[(X^*, Y^*, Z^*)]$.
    3.2 If $h_i = 1$ and $k_i = 0$ then set $S = S + P$
        $(X^*, Y^*, Z^*)$ = Add $[(X^*, Y^*, Z^*), (X, Y, Z)]$.
    3.3 If $h_i = 0$ and $k_i = 1$ then set $S = S - P$
        $(X^*, Y^*, Z^*)$ = Subtract $[(X^*, Y^*, Z^*), (X, Y, Z)]$.
Example: $k = 13$
$k = 001101,\ h = 100111 \to S = 2\{2[2(2P) - P]\} + P$
```

Figure 1: Point multiplication algorithm [8]

1.	2(2(2 S))	= 8 S + 0P
2.	2(2(2 S + P))	= 8 S + 4P
3.	2(2(2 S – P))	= 8 S – 4P
4.	2(2(2 S) + P)	= 8 S + 2P
5.	2(2(2 S + P) + P)	= 8 S + 6P
6.	2(2(2 S – P) + P)	= 8 S – 2P
7.	2(2(2 S) – P)	= 8 S – 2P
8.	2(2(2 S + P) – P)	= 8 S + 2P
9.	2(2(2 S – P) – P)	= 8 S – 6P
10.	2(2(2 S)) + P	= 8 S + P
11.	2(2(2 S + P)) + P	= 8 S + 5P
12.	2(2(2 S – P)) + P	= 8 S + 3P
13.	2(2(2 S) + P) + P	= 8 S + 3P
14.	2(2(2 S + P) + P) + P	= 8 S + 7P
15.	2(2(2 S – P) + P) + P	= 8 S – P
16.	2(2(2 S) – P) + P	= 8 S – P
17.	2(2(2 S + P) – P) + P	= 8 S + 3P
18.	2(2(2 S – P) – P) + P	= 8 S – 5P
19.	2(2(2 S)) – P	= 8 S – P
20.	2(2(2 S + P)) – P	= 8 S + 3P
21.	2(2(2 S – P)) – P	= 8 S – 5P
22.	2(2(2 S) + P) – P	= 8 S + P
23.	2(2(2 S + P) + P) – P	= 8 S + 5P
24.	2(2(2 S – P) + P) – P	= 8 S – 3P
25.	2(2(2 S) – P) – P	= 8 S – 3P
26.	2(2(2 S + P) – P) – P	= 8 S + P
27.	2(2(2 S – P) – P) – P	= 8 S – 7P

Figure 2: Point multiplication per 3 bits of scalar k

```
Input: An integer k and an elliptic curve point P = (X, Y, Z).
Output: The elliptic curve point S = k.P = (X*, Y*, Z*).
1. Set S = P
2. Let kl kl–1...k1 k0 and hl hl–1 ...h1 h0 be the binary
   representations of k and h=3k, respectively.
3. For i from (l – 1)/3  downto 1 do the following for every
   three bits of k and h
        3.1 S = (8 S ±mP)
        where mP is either P, 2P, …, or 7P depending of the
        bit pattern of k and h.
        (P, 2P, …, and 7P are pre-calculated and stored)
```

Figure 3: Modified point multiplication algorithm

the bit pattern of scalars *k* (and *h*), *8S* ± *mP* is performed for every three bits of the key. Notice that for the case of *8S*, a dummy addition *(8S + 0P)* can be performed in order to keep the datapath busy and in the end the result is exchanged with the value of *8S* which is already calculated. The modified algorithm is shown in Figure 3.

3.2. Performance optimization

Considering three bits of the integer *k* at a time not only helps to hide the key pattern from the attacker, but also will provide delay optimization in the overall point multiplication algorithm. The speed optimization is that there will be only one addition/subtraction for every three bits of integer k while the original algorithm requires one to three repeated additions/subtractions depending of the bit pattern of *k* and *h*. For example Figure 2 shows that in line 14, the original algorithm requires three doublings and three additions while the modified approach performs three doublings and only one addition. Therefore, the total point multiplication delay is minimized with the trade-off of having more memory to store the values *P, 2P, ..., 7P*.

A similar method was also presented in [10] at the algorithm level (not implementation) for the point multiplication algorithm based on the binary representation. However, in this paper we use the algorithm based on the signed digit representation and the hardware architecture of the proposed modified algorithm is presented. Moreover, we will explore further delay optimizations when performing *8S* ± *mP* as will be presented in the next section.

3.3. Point Double/Add/Subtract schedules

Figure 4 shows the details of the double and add/subtract operations based on the projective coordinate representation of the points of the elliptic curve [8], [11]. These operations are defined for the curve of $y^2 + xy = x^3 + a x^2 + b$ over $GF(2^n)$. The ECC operations are performed using multiplication, squaring, and addition in the underlying Galois Field $GF(2^n)$. To implement the high speed ECC operators based on the algorithms of Figure 4, the dataflow graphs of these operations and data dependencies between different variables and underlying Galois Field operators must be derived.

In order to minimize the overall delay of *8S* ± *mP* operation, the following two characteristics can be used.

- First of all, we can use multiple functional units (GF multiplier, squarer or adder) in parallel. For this purpose we should implement high speed and low cost (in terms of area) Galois field operators. In our case we have chosen the bit-serial implementation of the GF multiplier and squarer operations. This is because the bit-serial multiplier and squarer consume much less area than bit-parallel implementation and at the same time can be clocked at much faster clock frequencies due to the minimum combinatorial critical path delay.
- Secondly, since we are calculating *8S* in the first phase we can combine the operation schedules of three doubles in a row together and optimize the total delay. Moreover, due to the fact that we can use more than one GF multiplier and/or squarer, by maintaining the data dependencies between different operators in the schedule, we can move the GF operator up or down in the schedule and therefore find the optimal schedule and operator assignments that can perform the *8S* ± *mP* operation.

Figure 5 shows the optimized schedule of three double operations (*8S*) that is derived using two multipliers, one squarer and one adder. In this dataflow graph, the box with sign × is the GF multiplier, the box with sign 2 is the GF squarer, and the box with sign + is the GF adder.

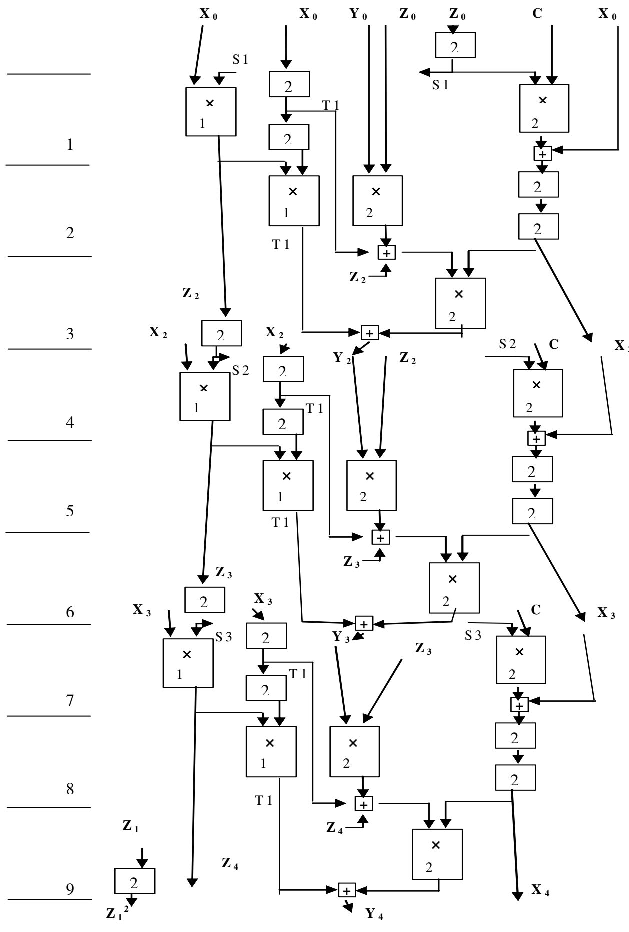
$2\,(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$, where
$Z_2 = X_1\, Z_1^2$,
$X_2 = (X_1 + c\, Z_1^2)^4$,
$U = Z_2 + X_1^2 + Y_1 Z_1$,
$Y_2 = X_1^4\, Z_2 + U X_2$.

(a) Double operation

$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$, where
$U_0 = X_0\, Z_1^2$,
$S_0 = Y_0\, Z_1^3$,
$U_1 = X_1\, Z_0^2$,
$W = U_0 + U_1$,
$S_1 = Y_1\, Z_0^3$,
$R = S_0 + S_1$,
$L = Z_0\, W$
$V = R X_1 + L Y_1$,
$Z_2 = L Z_1$,
$T = R + Z_2$,
$X_2 = a\, Z_2^2 + TR + W^3$,
$Y_2 = T X_2 + V L^2$.

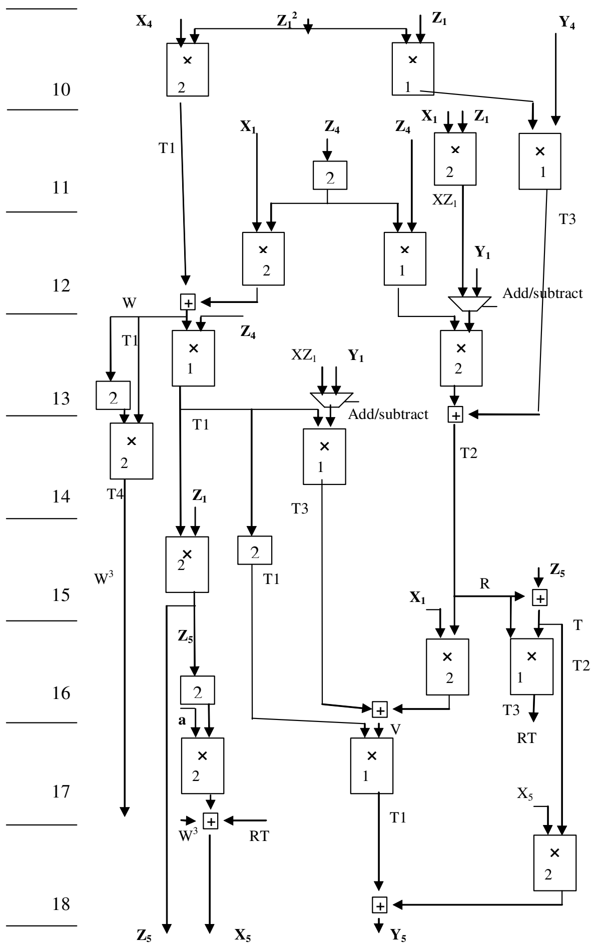
(b) Add/Subtract operation

Figure 4: Double and Add/Subtract algorithms



$S = (X_0, Y_0, Z_0)$
$2S = (X_2, Y_2, Z_2) = 2(X_0, Y_0, Z_0)$
$4S = (X_3, Y_3, Z_3) = 2(X_2, Y_2, Z_2)$
$8S = (X_4, Y_4, Z_4) = 2(X_3, Y_3, Z_3)$
Each doubling is based on the dataflow graph of Figure 4a

Figure 5: Optimized schedule for 3-Double operations



$8S = (X_4, Y_4, Z_4)$
$mP= (X_1, Y_1, Z_1)$
$8S \pm mP = (X_5, Y_5, Z_5)$.
Add/subtract is based on the equations of Figure 4b

Figure 6: Optimized schedule for add/subtract operations

In Figure 5, the input point (S) is (X_0, Y_0, Z_0) and the output $(8S)$ is (X_4, Y_4, Z_4), and the three double operations based on the data flow graph of Figure 4 are scheduled in a sequence. As it will be shown in the next section, the $GF(2^n)$ bit-serial multiplier and squarer require $(n+1)$ and $(n+1)/2$ clock cycles to generate their outputs, respectively. This means that we can perform two GF squarings in the time frame of a GF multiplication. Therefore, for every time frame that is numbered in Figure 5, two GF multiplications and two GF squarings are performed using the two multipliers and the single squarer. The GF adder takes only one cycle and is performed in the end of each time frame. By considering another single cycle to store the result in the memory and load the new operands, there will be total of $(n+3)$ cycles required to perform each step.

Figure 6 shows the optimized schedule of the add/subtract operation ($8S \pm mP$). The inputs are of Figure 5, $8S=(X_4, Y_4, Z_4)$ and $mP=(X_1, Y_1, Z_1)$ which is loaded from memory and the final result is (X_5, Y_5, Z_5). A control signal differentiates between add and subtract operations by loading the appropriate operands. This is because subtraction of (X_4, Y_4, Z_4) - (X_1, Y_1, Z_1) is calculated by addition of (X_4, Y_4, Z_4) + $(X_1, X_1 Z_1 + Y_1, Z_1)$ [8].

Using the schedule of figure 5 the calculation of $8S$ takes $[9(n+3) + (n+3)/2]$ cycles which is $6(n+3)$ cycles less than the original algorithm of Figure 4a. Moreover the schedule of figure 6 takes $9(n+3)$ cycles which *is* $[7(n+3) + (n+3)/2]$ cycles less than the original implementation of $8S \pm mP$ using the algorithm of figure 4b.

3.4. Underlying Galois Field operations

This section presents the architecture of underlying Galois Field operators. The configurable and scalable architecture of general GF(2^n) operators are used. The multiplier and squarer are performed modulo an irreducible polynomial (P) which is a programmable parameter. The bit-serial implementations of these operators are used. As an example, the architecture of a 4-bit multiplier is shown in Figure 7. Generally there are total of n registers that contain the output R. It takes one cycle to load the operands A and B followed by n cycles of calculating the product and modulo reduction [12]. Therefore, the result of multiplication is ready after $(n+1)$ cycles. Figure 8 shows the architecture of the bit-serial multiplier (a 7-bit field is shown as an example). Squaring is similar to the multiplication with the difference being the two operands are the same. Therefore, half of the bits of the input can be loaded into the output registers and every new coefficient is shifted over two positions [12]. Therefore, for an n bit squarer, a total of (n+1)/2 cycles (half of the multiplication) are required. Notice that the architecture shown in Figure 8 is for the case when n is odd. When n is even there is a slight difference in the connection of the irreducible polynomial P and the result registers. We have chosen the odd case because for secure elliptic curve cryptography based on GF(2^n), n must be a prime number.

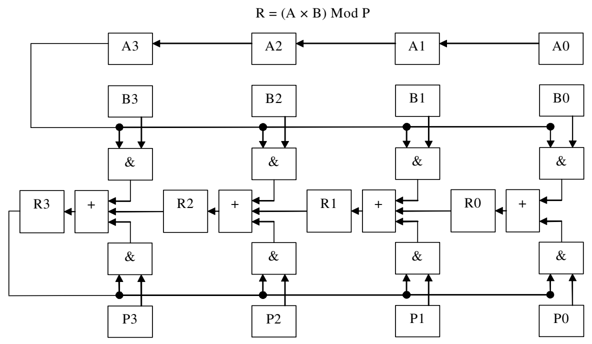


Figure 7: GF(2^n) bit-serial multiplier

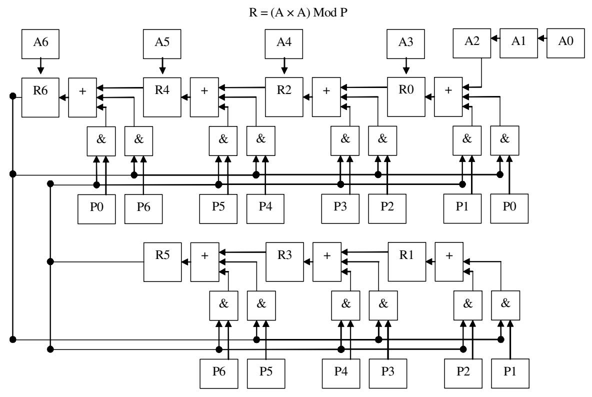


Figure 8: GF(2^n) bit-serial squarer

The GF adder chosen for our implementation is a bit-parallel architecture in order to generate the result in a single clock cycle. Notice that the addition in GF(2^n) is carry free. Therefore, addition is equivalent to XORing the two bit-vectors of the input operand.

4. Processor architecture

Based on the modified point multiplication algorithm and the optimized schedules of elliptic curve three doubles and add/subtract operations that are presented in section three, a scalable and high speed elliptic curve cryptographic processor is implemented. The underlying Galois Field operations that are presented in the last section are used in the datapath. Based on the schedules of Figures 5 and 6 the required interconnection between storage elements and GF operators are designed and the finite state machine that follows the schedules is implemented.

Figure 9 shows the point multiplication datapath that is designed to calculate $8S \pm mP$ schedules. This module includes a datapath that consists of the GF operators with their interconnections, the storage unit that keeps the intermediate variables (*X, Y, Z, T1, T2, T3, T4*), and the FSM that creates the control signals to perform the $8S \pm mP$ operation schedule (Figures 5 and 6). Note that point mP with coordinate (X_1, Y_1 , Z_1) is an input to this module. The result of each doubling and the final addition/subtraction are overwritten to the initial register variable S. This means that the hardware registers that store X_0 , X_2 , X_3 , X_4 , and X_5 are the same. This is also the case for Y and Z coordinates. Moreover, four temporary variables called *T1, T2, T3, T4* are used to store the values in the whole schedule.

The point multiplication module of figure 9 is controlled from the upper level module using the *start* and *done* and *plus_minus* signals. Basically, the upper level control provides the correct value of mP through (X_1, Y_1, Z_1) connections and also asserts the write value for *plus_minus* signal and asserts the *start* signal of this unit. Then this unit that already has the value of S in its storage (variables *X, Y, Z)* starts to calculate $8S \pm mP$ and updates the variables *X, Y, Z* and asserts the *done* signal indicating that the result for the three bits is ready. This process is repeated for every three bits of k.

Figure 10 shows the block diagram of the whole processor. The ECC storage unit stores the values mP. The key scheduling unit calculates the value $h = 3k$ and generates the decision signal for the ECC storage unit to choose the value mP for every three bits of k and h. The top level controller issues the required controls for the point multiplication datapath and the key scheduling unit to synchronize the operation of all the units in the processor.

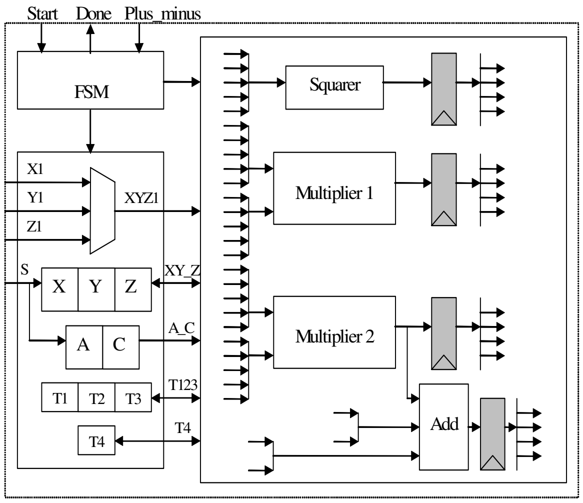


Figure 9: Point multiplication Datapath

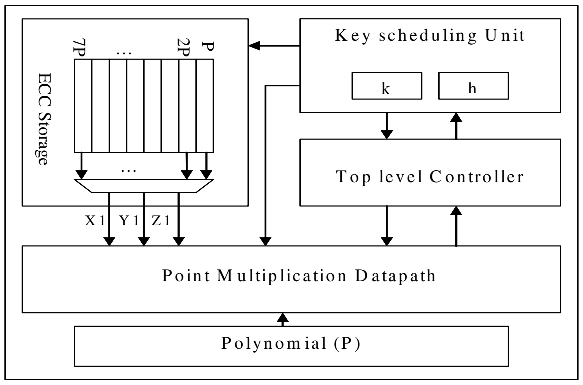


Figure 10: Processor Architecture

5. Performance Results

The proposed elliptic curve crypto processor is designed using VHDL and is simulated using Modelsim. This processor is scalable and can be generated for any field $GF(2^n)$ where n is the size of the datapath. Every time frame of the schedules in Figures 5 and 6 takes $(n+3)$ cycles. This means that the whole $8S \pm mP$ schedule takes $[18(n+3) + (n+3)/2 + 1]$ cycles where $18(n+3)$ is for the 18 time frames and $(n+3)/2$ is for the first square in figure 5 and the last single cycle is to update the registers with the final result. Since the $8S \pm mP$ operation is repeated for every three bits of scalar k and the maximum size of scalar k is n bits long; therefore, the point multiplication will take $[18(n+3) + (n+3)/2 + 1]\times(n/3)$ clock cycles. For the case of GF(2^{163}), which is considered to be as secure as 1024-bit RSA, the $8S \pm mP$ operation takes total of 3072 cycles. Therefore, the delay of point multiplication in GF(2^{163}) is 165888 clock cycles.

6. Conclusion

A high performance and scalable elliptic curve processor that provides resistance against timing attacks is presented. The dataflow schedules for the underlying operators of the modified point multiplication algorithm are optimized for maximum speed. The architecture of the proposed processor is based on the Galois Field of $GF(2^n)$ and is configurable for any value of n. The total delay of point multiplication is $[18(n+3) + (n+3)/2 + 1]\times(n/3)$ clock cycles. For the case of GF(2^{163}) the point multiplication delay is 165888 clock cycles.

7. Acknowledgement

This research has been supported by the Space and Naval Warfare Systems Center - San Diego under contract No. N66001-02-1-8938.

8. References

[1] Nils Gura, Arun Patel, Arvinderpal Wander, et al. “Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs”, *CHES 2004*, pp 119-132.

[2] J-S. Coron, “Resistance against differential power analysis for elliptic curve cryptosystems,” *CHES 1999*, LNCS 1717, pp. 292-302, 1999.

[3] J. Lopez, R. Dahab, “Fast multiplication on elliptic curves over GF(2m) without precomputation” *CHES 1999*, LNCS 1717, pp. 316-327, 1999.

[4] K. Okeya, K. Sakurai, “Power analysis breaks elliptic curve cryptosystems even secure against the timing attack,” *Indocrypt2000*, LNCS1977, pp. 178-190, 2000.

[4] E. Oswald, M. Aigner, “Randomized addition-subtraction chains as a countermeasure against power attacks,” *CHES 2001*, LNCS 2162, pp. 39-50, 2001.

[6] E. Trichina, A. Bellezza, “Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks,” *CHES 2003*, LNCS 2523, pp. 98-113, 2003.

[7] B. Moller, “Securing Elliptic Curve Point Multilication against Side-Channel Attackes” *Proc. Information Security 2001*, LNCS 2200, pp. 324-334, 2001.

[8] IEEE P1363/D13, Standard Specification for Public-key Cryptography, November 1999.

[9] L. Batina, S. Berna, B. Preneel, J. Vandewalle, “Hardware architectures for public-key cryptography,” *Elsevier Integration the VLSI Journal*, vol. 34, pp. 1-64, 2003.

[10] D. Gordon,“A survey of fast exponentiation methods,” *Journal of Algorithms*, vol.27, pp.129-146, 1998.

[11] E. Win, B. Preneel, “Elliptic curve public-Key cryptosystems—an introduction,” LNCS 1528, pp. 131-141, 1998.

[12] S. Janssens *et al.*, “Hardware/software co-design of an elliptic curve public-key cryptosystem,” Proc. SIPS, pp. 209-216, 2001.A.B. Smith, C.D. Jones, and E.F. Roberts, “Article Title”, *Journal*, Publisher, Location, Date, pp. 1-10.