SPA resistant Elliptic Curve Cryptosystem using Addition Chains

A. Byrne*, N. Meloni†, F. Crowe*, W.P.Marnane*, A. Tisserand† and E.M.Popovici‡
*Dept. of Electrical and Electronic Engineering, University College Cork
Email: {andrewb,liam,francisc}@rennes.ucc.ie
†LIRMM, CNRS - Univ. Montpellier 2
Email: {nicolas.meloni,arnaud.tisserand}@lirmm.fr
‡Dept. of Microelectronic Engineering, University College Cork
Email: {e.popovici}@ucc.ie

***Abstract*— There has been a lot of interest in recent years in the problems faced by cryptosystems due to side channel attacks. Algorithms for elliptic curve point scalar multiplication such as the *double and add* method are prone to such attacks. By making use of special addition chains, it is possible to implement a Simple Power Analysis(SPA) resistant cryptosystem. In this paper a reconfigurable architecture for a cryptographic processor is presented. A SPA resistant algorithm for point multiplication is implemented and is shown to be faster than the *double-and-add* method. Post place and route results for the processor are given.**

Keywords - Cryptography, elliptic curves, reconfigurable architecture, addition chains, side-channel attacks

I. Introduction

Elliptic curve cryptography was proposed by Miller[1] and Koblitz[2] in 1985. It provides a means for two hosts to generate a secret key for communication across an insecure channel. The strength of cryptography lies in the difficulty of an encryption schemes inverse operation. Elliptic curve cryptography provides relatively better security per bit than other cryptographic standards such as RSA.

Therefore elliptic curve cryptosystems (ECC) consume less memory and hardware resources to implement. The main operation of ECC is scalar point multiplication given an elliptic curve E and a point P on E the point $[k]P = P+P+\dots P$ for some given integer k. The basis for the strength of the ECC is the *elliptic curve discrete logarithm problem* (ECDLP). Given two points Q and P on an elliptic curve E, find the integer k such that $Q = kP$. This is the operation required to retrieve a secret key from an ECC. For a large enough key size, a brute force attack would require too much computing power and time to be feasible[3].

Recently, more effort has been carried out to secure EC point multiplication against side channel attacks[4]. SPA attacks monitor the power consumption of an execution of a cryptographic algorithm. Algorithms such as the *double-and-add* method are prone to these types of attacks. Euclid's addition chains can provide both a secure and efficient scheme of exponentiation when combined with elliptic curves[5].

The remainder of the paper is structured as follows: Section II provides some background information on elliptic curves. Section III briefly introduces Side Channels Attacks. Section IV presents addition chain theory. Section V describes the versatile processor and all it's components. Implementation results are given.

II. Elliptic Curves

An elliptic curve $E(GF(q))$ over GF(q) is the set of points $P = (x, y)$, $x, y \in GF(q)$ such that

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,\ a_i \in GF(q) \quad (1)$$

along with a special point at infinity ∂.

Elliptic curves over large prime fields are described using the Weierstrass equation

$$y^2 = x^3 + a_4x + a_6 \quad (2)$$

where x, y, a_4 and $a_6 \in GF(q)$ and $4a_4^3 + 27a_6^2 \neq 0$.

The number of points on the curve is $p + 1 - t$, where t is the trace of Frobenius(Tr). Given a point $P = (x, y)$ and a positive integer n, the order of P is the smallest positive integer n such that $[n]P = \partial$. The curve Q where t is a multiple of p is *supersingular*, otherwise it is *non-supersingular*. The choice of curve determines the group law and defines the equations for point doubling and point addition.

Points on the elliptic curve can be represented in Jacobian coordinates which avoids the need for an expensive inversion operation[6]. Converting from affine to projective coordinates is a simple operation, $(x, y, 1) \rightarrow (X, Y, Z)$. Conversion back however requires a number of modular multiplications and inversions, $(x, y) \leftarrow (\frac{X}{Z^2}, \frac{Y}{Z^3})$. In Jacobian coordinates, the curve in Equation 2 is given by $Y^2 = X^3 + a_4XZ^4 + a_6z^6$

A. EC Multiplication

Given an elliptic curve E and a point P on the curve, the point Q is calculated by point scalar multiplication where the point P is added to itself k times to get the point $[k]P$.

Algorithms such as the *double and add* algorithm, given in Algorithm 1 are used to calculate point multiplication efficiently. The *double and add* algorithm requires n_k point doubling operations (n_k is the bit length of the key) and $w(k)$ point additions ($w(k)$ is the binary weight of the key

International Conference on Information Technology (ITNG'07)
0-7695-2776-0/07 $20.00 © 2007 IEEE

Algorithm 1: Double and Add Point Scalar Multiplication

input : $P \in E(GF(q)), k = \sum_{i=0}^{n_{k-1}} k_i 2^i$
output: $Q = [k]P \in E(GF(q))$
Initialise: Q=P;
for $i \leftarrow n_k - 2$ **to** 0 **do**
  $Q = 2Q$ //Point Doubling;
  if $k_i = 1$**then**
    $Q = Q + P$ *//Point Addition;*
  end
end

B. EC Point Addition and Doubling

Consider two separate points on an elliptic curve, $P = (x_p, y_p)$ and $Q = (q_t, q_t)$. A line l is drawn through the points P and Q. The line l intersects the curve at a third point, $Q' = (x_{q'}, y_{q'})$ is the inverse of that point, where $Q' = P+Q$. The point addition formulae for the curve defined in Equation 2 using Jacobian coordinates are given in Algorithm 2. The computational cost of a point addition is 15 multiplications and 7 add/subs.

Algorithm 2: Point Addition in Jacobian Coordinates

input : $P(X_1, Y_1, Z_1), Q(X_2, Y_2, Z_2) \in GF(q)$
output: $P + Q(X_3, Y_3, Z_3) \in E(GF(q))$
$A = X_1 Z_2^2$, $B = X_2 Z_1^2$, $C = Y_1 Z_2^3$, $D = Y_2 Z_1^3$;
$E = B - A$, $F = D - C$;
$X_3 = -E^3 - 2AE^2 + F$;
$Y_3 = -CE^3 + F(AE^2 - X_3)$, $Z_3 = Z_1 Z_2 E$

If $T = P$ then this is *point doubling* and a tangent to the point is used. The tangent intersects with the curve at a second point, $T' = 2(T)$ is the inverse of this point. Algorithm 3 gives the formulae for point doubling for the curve in Equation 2.The computational cost for point doubling is 10 multiplications and 8 add/subs.

Algorithm 3: Point Doubling in Jacobian Coordinates

input : $P(X_1, Y_1, Z_1) \in GF(q)$
output: $[2]P(X_3, Y_3, Z_3) \in E(GF(q))$
$A = 4X_1 Y_1^2$, $B = 3X_1^2 + a_4 Z_1^4$;
$X_3 = -2A + B^2$;
$Y_3 = -8Y_1^4 + B(A - X_3)$, $Z_3 = 2Y_1 Z_1$

III. Side Channel Attacks

In recent years, cryptosystems have come under attack from various forms of side channel attack. Kocher *et al.*[4] discovered that cryptosystem implementations leak information which can help an attackers recover secret data. One such technique for retrieving secret information is SPA.

SPA involves monitoring the power consumption of a single execution of a cryptographic algorithm. Every instruction has a different power consumption, therefore it is possible to retrieve the sequence of instructions during the algorithm execution. For example, the *double and add* algorithm has two primary operations, point addition and point doubling. Each of these operations produce a different power trace when executed because of the different number of multiplications and additions in each algorithm. Since, the execution of a point addition in the *double and add* is directly related to the secret key, it is possible to retrieve the secret key by monitoring the power consumption of a single execution of a scalar multiplication. The first successful power analysis attack against an FPGA was done by Ors *et al.*[7] in which they attacked an elliptic curve processor and retrived the secret key.

SPA attacks work well on algorithms where the the power consumption can be directly related to the instruction being executed. In order to resist SPA attacks, the instructions executed in a cryptographic algorithm must not be directly related to the secret data. In the double and add method, the branch instruction based on k_i leaks information about the secret key. A simple solution would be to execute a point doubling for every bit of k but this vastly increases the execution time of the algorithm. In this paper, we make use of special *Addition Chains* to perform a point multiplication using only point additions. In this way, SPA cannot be used to determine the secret key.

IV. Euclid's Addition Chains

An addition chain is a finite sequence of integers $(v_0, \ldots, v_s)$ satisfying $\forall k \leq s, v_k = v_i + v_j$ for some $i, j < k$. An Euclid's addition chains is an addition chain which satisfies $v_1 = 1, v_2 = 2, v_3 = v_2 + v_1$ and $\forall\ 3 \leq i \leq s-1$, if $v_i = v_{i-1} + v_j$ for some $j < i - 1$, then $v_{i+1} = v_i + v_{i-1}$(case 1) or $v_{i+1} = v_i + v_j$(case2).

Case 1 is called the Fibonacci step (it corresponds to one step of the Fibonacci sequence) and case 2 is called a small step (we add the smallest of the two possible numbers to v_i).

As an example, $(1, 2, 3, 4, 7, 11, 15, 19, 34)$ is an Euclid's addition chain computing 34, for instance in step 4 we have computed 4=3+1, thus in step 5 we must add 3 or 1 to 4, in other words from step 4 we can only compute 5=4+1 or 7=4+3. In this example we have chosen to compute 7=4+3 so that at step 6 we can now compute 10=7+3 or 11=7+4 etc.

Finding such chains is quite simple, it suffices to choose an integer k' coprime with k and apply the subtractive form of Euclid's algorithm. Let $k = 34$ and $k' = 19$ and apply the subtractive form of Euclid's algorithm:

$$
\begin{aligned}
34 - 19 &= 15 && \text{(Fibonacci step)}\\
19 - 15 &= 4 && \text{(small step)}\\
15 - 4 &= 11 && \text{(small step)}\\
11 - 4 &= 7 && \text{(Fibonacci step)}\\
7 - 4 &= 3 && \text{(Fibonacci step)}\\
4 - 3 &= 1 && \text{(small step)}\\
3 - 1 &= 2\\
2 - 1 &= 1\\
1 - 1 &= 0
\end{aligned}
$$

The main advantage of these chains[5] is that they're only made of additions (no doublings) which makes them resistant against simple channel analysis. In order to adapt thos

to the elliptic curve point multiplication, we have introduced new addition point formulae (in the case of elliptic curve over prime field) taking advantage of the particular structure of the chains. Their computational cost is 5 field multiplications and 2 field squares as shown in Section IV-A.

Up to now Euclid's chain was mainly used with curves in Montgomery form (on which the addition of two points P and Q can be performed in only 4 multiplications and 2 squarings if the difference P-Q is known), however not every curve can be transformed into Montgomery form (moreover those curves are not of prime order and do not satisfy the NIST recommendations). On the other hand, Brier and Joye proposed Montgomery like formulae working with any curve but in this case the addition cost is 9 multiplications and 2 squarings presenting a gain of 37%.

The drawback of Euclid's chains is the fact that it is not easy to find small ones. Several methods have been proposed to speed up the computation of small chains but in the end it still remains a "clever" exhaustive search. The following table summarizes the number of attempts to find an Euclid's chain of a given length:

chain length	320	300	280	260
on average	29	121	2353	7,795,840
worst case	521	3,454	44,254	79,402,210

TABLE I
NUMBER OF ITERATIONS NEEDED TO FIND A EUCLID'S ADDITION CHAIN COMPUTING A 160 BITS INTEGER

A. New Elliptic Curve Formulae

Given a point P on the elliptic curve E, an integer k and $v = (v_3, ..., v_s)$, a special addition chain computing k and s the addition chain length. In order to simplify the algorithm, we will use the following notation : if $v = (1, 2, 3, v_3, \ldots, v_s)$ is an EAC then we only consider the chain from v_3 and we replace all the v_i's by 0 if it has been computed using a Fibonacci step and by 1 for a small step.

Now it is easy to deduce the point scalar multiplication algorithm in Algorithm 4.

Algorithm 4: Point Scalar Multiplication using Addition Chains

input : $P \in E(GF(q)), k = (v_3, ..., v_s)$
output: $Q = [k]P \in E(GF(q))$
Initialise: $U_1, U_2, U_3 \leftarrow (P, [2]P, [3]P)$;
for $i \leftarrow 3$ **to** s **do**
  if $w_i = 0$ **then**
    $U_1 = U_2$;
  end
  $U_2 = U_3$;
  $U_3 = U_1 + U_2$;
end

From Algorithm 4 we need to perform one initial point doubling followed by $s - 1$ point additions.

If P and Q share the same Z-coordinate we can reduce the point addition and doubling formulae. The new point addition formulae are given in Algorithm 5. Using this formulae, the computational cost is greatly reduced to 7 multiplications and 7 add/subs. One can verify that the output point P' is equivalent to P in jacobian coordinates, so that a common Z-coordinate can be maintained between the added points all along the algorithm.

Algorithm 5: Point Addition, P and Q sharing same Z coordinate

input : $P(X_1, Y_1, Z), Q(X_2, Y_2, Z) \in GF(q)$
output: $P + Q(X_3, Y_3, Z_3), P'(X_1', Y_1', Z_3) \in E(GF(q))$
$A = (X_2 - X_1)^2, X_1' = X_1 A, B = X_2 A, D = (Y_2 - Y_1)^2$;
$Y_1' = Y_1(B - X_1')$;
$X_3 = D - X_1' - B$;
$Y_3 = (Y_2 - Y_1)(X_1' - X_3) - Y_1'$, $Z_3 = Z(X_2 - X_1)$

V. ELLIPTIC CURVE PROCESSOR

A generic architecture was designed for cryptographic operations which incorporates RAM, a ROM controller and a number of arithmetic units for a given field. The processor can also be configured to perform pairing operations. Software was developed using C++ to generate the VHDL for a customized processor for any characteristic *p* and extension field *m*. Everything from the size of the RAM block to configuring the arithmetic units and generating the ROM instruction set for a given algorithm is controlled by the program.

For prime characteristic fields there is a choice of arithmetic units to chose from for the architecture. Through manipulation of the ROM instructions alone, the processor can be configured for various algorithms including the *double-and-add* algorithm or exponentiation using addition chains. In this way, we can quickly compare these and other cryptographic algorithms. In the next section we will look at the arithmetic units used in the processor.

A. Arithmetic Units

The point addition and doubling algorithms described in Sections II-B and IV-A require modular additions, subtractions and multiplications. While addition and subtraction are relatively easy to implement, modular multiplication is much more complex. An in depth review of modular arithmetic and architectures can be found in [8]

The processor architecture in Figure 1 is capable of controlling a number a arithmetic units. There are two architecture types available for the processor. The first is a configurable arithmetic logic unit (ALU) that can be set to perform modular addition, subtraction or multiplication. Taking under consideration the speed/area constraints of the target technology and the application of the design, the number of ALUs can be changed to give optimum results. Alternatively, we can use dedicated units for each of addition, subtraction and multiplication. Since addition and subtraction only take 4 clock cycles to complete, two of which are RAM read/writes, these operations are best performed in series and do not gain from an increased number of arithmetic units. Therefore by using dedicated units, we can increase the number of modular multipliers only and save area.

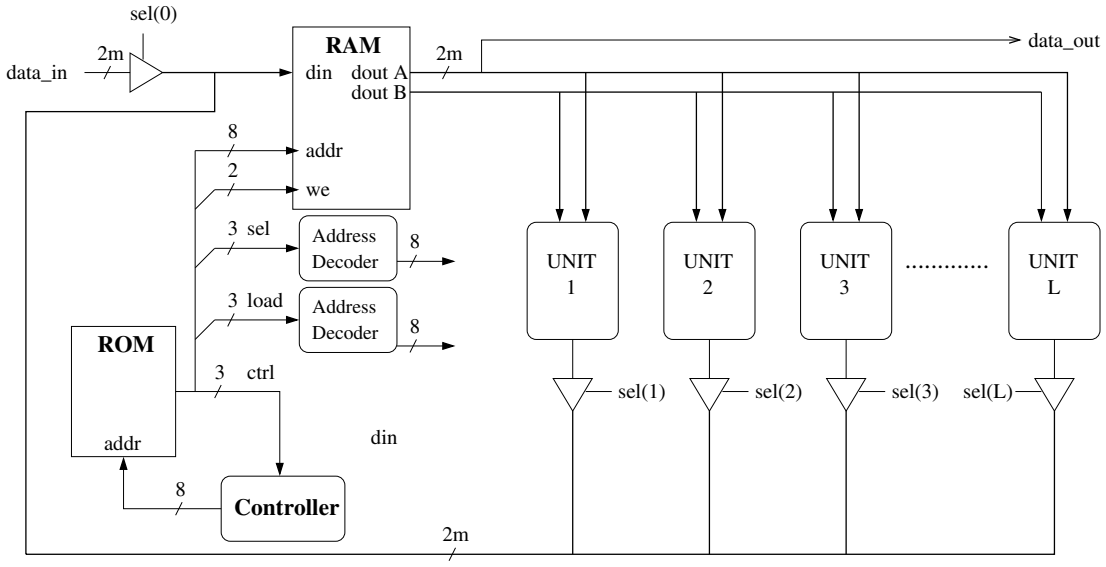


Fig. 1. General Elliptic Curve Processor

1) Multiplication: In 1985 Montgomery[9] proposed an efficient method for performing modular multiplication using a series of additions and right shifts. This method avoids the need for costly trial division of the modulus. The Montgomery modular product is defined in Equation 3.

$$Res = Mont(A, B, p) = AB2^{-p_b+2} \,(mod\, p) \quad (3)$$

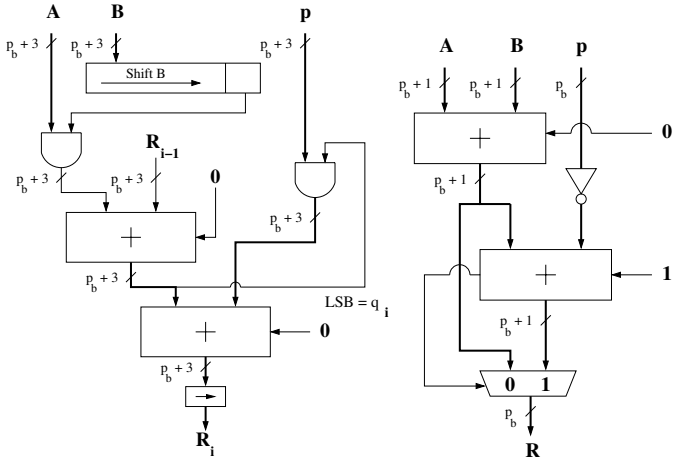
The output of a Montgomery multiplication is a factor 2^{-p_b+2} times smaller than the desired result, p_b is the field size in bits. In order to correct the result, the output must be Montgomery multiplied by $(2^{2p_b+2} \; mod\, p)$. When a large number of multiplications are required it becomes inefficient to correct every result. A better solution is to initially convert the numbers to the Montgomery domain. To do this, the number is Montgomery multiplied by $(2^{2p_b+2} \; mod\; p)$. To convert a number back, it is Montgomery multiplied by 1.

The algorithm for the Montgomery multiplication is given in Algorithm 6. The number of iterations performed is $p_b + 2$ in order to the bound the output in the range $[0, 2p - 1]$ for multiplicands up to twice the modulus. This allows it to be used as an input to further multiplications without the need for conditional subtraction.

Algorithm 6: Montgomery Multiplication

input : $A = \sum_{i=0}^{p_b} a_i 2^i$, $B = \sum_{i=0}^{p_b} b_i 2^i$,;
$M = \sum_{i=0}^{p_b} p_i 2^i$
output: $R = AB2^{-p_b+2} \,(mod\, p)$
Initialise: $R \leftarrow 0$; $b_{p_b+1} \leftarrow 0$;
for $i \leftarrow 0$ **to** $p_b + 1$ **do**
  $q_i = R_{i-1} + b_i A \,(mod 2)$;
  $R_i = (R_{i-1} + q_i M + b_i A)/2$;
end

A hardware implementation of the Montgomery multiplier can be seen in Figure 2(a). Multiplication is performed according to Algorithm 6. The inputs to the first adder are $b_i A$ and the previous result R_{i-1}. $q_i p$ is added to the sum of the first adder if the LSB of the sum (q_i) is equal to 1. A shift register scans each bit of B for $b_i A$ and the final result is right shift divided by 2.



(a) Multiplier (b) Adder

Fig. 2. Modular Adder and Multiplier

2) Modular Addition and Subtraction: The modular addition operation adds A and B in the first adder and subtracts the modulus p from the sum. To subtract the modulus from the intermediate result, the modulus is bitwise inverted and added to $(A + B)$ with the carry-in set to 1, thus performing a two's complement subtraction. The carry-out of the second adder controls which intermediate result is the correct result. If $(A + B)$ is in the correct range, the result of the first adder is the correct result. Otherwise, the result from the second adder is correct. The architecture for the adder/subtractor in Figure 2(b) is configured for modular addition.

Modular subtraction is performed similarly. In this case however, B is bitwise inverted and added to A with the carry in set to 1. If the carry-out of this adder is low, the modulus must be added to give an output in the correct range.

3) Configurable ALU: Figure 3 shows the architecture for the configurable ALU. The modular addition, subtraction and multiplication operations are controlled with a 2-bit *mode* signal set to 00,01 and 10 for these operations respectively. A 2-bit load signal is used also to load the operands.

Table II gives the speed and area performance of the arithmetic units used for the processor. All results are based an 192-bit prime modulus.

	Configurable ALU	Multiplier	Adder/Subtractor
Slices	746	503	394
min. period (ns)	22.437	18.686	27.808
Clock Freq.	44.569Mhz	53.516Mhz	35.961Mhz

TABLE II

POST PLACE AND ROUTE RESULTS FOR XILINX XC2VP125

The dedicated multiplier, adder and subtractor give better area results than the configurable ALU. This is due to the extra control logic surrounding the ALU for selecting the mode thus reducing the minimum area. The Multiplier also performs considerably faster that the configurable ALU.

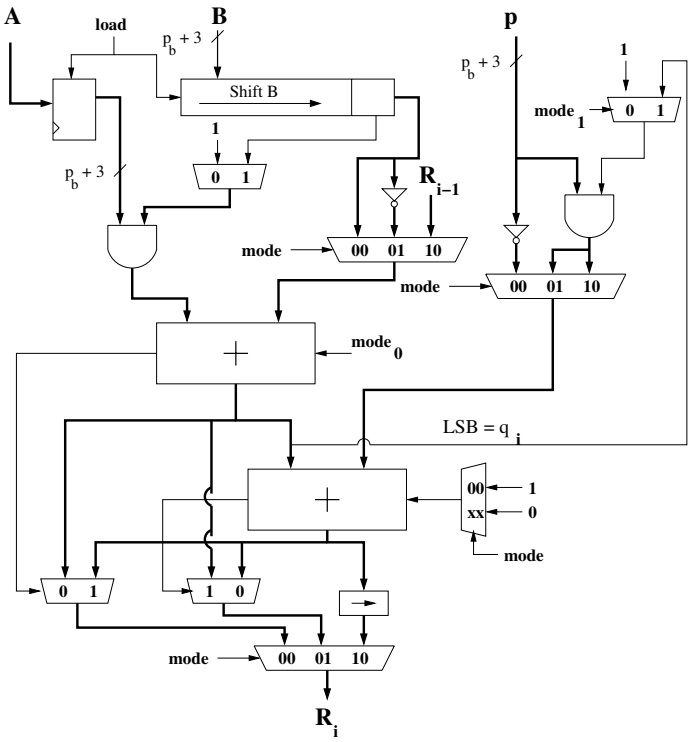


Fig. 3. Configurable ALU

B. ROM Instruction Set

When deciding the control for the processor there are two options. The first is a finite state machine which can be set up to perform specific operations such as elliptic curve point scalar multiplication or Tate pairings. This does not allow for much flexibility in the design however. Instead, the use of microcode stored in Xilinx BlockROM was implemented. A similar approach was taken by Leong *et al.*[10] which helped reduce the development time of the processor and increased the flexibility of the design. A major advantage of this is that the instruction set can be updated to perform any number of operations without the need to recompile the entire processor.

When generating the ROM instruction set several consideration need to be taken into account such as, are the points on the elliptic curve being represented by projective or affine coordinates; what algorithm is being implemented; how many arithmetic units are available and what are their timing constraints.

For the field p, p large prime, configurable and dedicated ALUs for modular addition, subtraction and multiplication based on [11] were implemented. In this section we will look only at the architecture implementing configurable ALUs. For the architecture to accommodate this type of unit, *mode* bits were needed to set the operation of the ALUs. The instruction set for a reconfigurable ALU based processor over GF(p) using projective coordinates is shown in Table III. After initially loading the elliptic curve parameters and Montgomery constants into RAM, the controller performs operations for the selected cryptographic algorithm.

Instruction Set					
ctrl	mode	load	sel	we	addr A & B
00	00000000	000	000	00	00001 00010
01	00100000	011	000	00	00000 00000
00	00000000	000	011	01	00000 00011

TABLE III

INSTRUCTION SET USING RECONFIGURABLE ALUS

Here we are using projective coordinates to represent the points on the elliptic curve to remove the need for inversions which are time consuming. The 12 LSBs control read and write access to the dual port RAM. Bits $12 \rightarrow 14$ control the tri-states connected to the outputs of the ALUs. Only one of these is set high when writing data to RAM. To reduce the impact of a large number of arithmetic units in the design on the size of ROM, a 3-to-8 address decoder is used to make full use of all combinations of the 3 select bits. Bits $15 \rightarrow 17$ are the load bits which are used to load new vectors to a specific ALU. Bits $18 \rightarrow 25$ control the mode signal for each ALU. In this example there are 4 ALUs in the design, 2 mode bits per ALU. The two MSBs are extra control bits for the state machine controlling the processor.

Some operations such as addition & subtraction execute in a single clock and have no extra timing requirements associated with them. Multiplication however takes $p_b + 3$ clocks. The controller for the ROM is a simple counter that goes through each address sequentially on each rising clock edge. To account for the operational time between loading the operands and getting the result a state machine is needed to handle these exceptions. By monitoring bit 26 of the instruction, the state machine can halt normal execution of the instructions for a set number of clock cycles while a series of multiplications are performed.

C. Hardware Generator

The versatile processor is presented in Figure 1. VHDL for the processor is generated by software developed in C++. This allows for the system to be completely reconfigurable for any characteristic and extension field. The generation of the ROM is explained in Section V-B. The architecture presented makes use of dual-port RAM(though it can be configured for single-port RAM) so both operands of an arithmetic unit can be loaded in parallel.

The processor in Figure 1 can be configured for a number of different algorithms over GF(p) using the arithmetic units described in Section V-A. It can also be configured for GF(2^m) and GF(3^m) where dedicated units are used for multiplication, addition, subtraction, inversion and division[12]. For GF(p) the number of configurable ALUs implemented can be modified using the software developed and is restricted by the FPGA resources and the point where additional units no longer give a substantial improvement in the design. Likewise, using dedicated units, the number of multipliers, adders and subtractors can be configured within the constraints of the target device and performance gain. All changes to the processor are handled by the software.

There is a trade off between speed and area that is affected by the number of ALUs implemented in the processor. More ALUs will reduce execution time but will increase the area consumption of the device. Schedules for each algorithm were generated for a varying number of ALUs and it was found that for the *double and add* algorithm, the best speed/area trade off was 4 ALUs. Given the reduced algorithm for point addition, there are only ever at most two multiplications executing in parallel. Therefore, there is no advantage gained in having extra ALUs. Using Algorithm 5 for point addition, the best schedule for a speed/area tradeoff implemented 2 ALUs. This processor gives a slight improvement in the maximum clock frequency but the total area of the device is half that of the *double and add* processor implementing 4 ALUs.

The architecture presented in Section V was evaluated on Xilinx xc2vp125. The post place and route results for point multiplication using the *double and add* and addition chain methods are listed in Table IV. Each design consumes approximately 5% of the Block RAMs available. The largest designs using 4 ALUs only need 28 bits for the ROM instruction set leaving room to extend the instruction for future applications. The results are based on a 160-bit key size. For the addition chains, a chain of length 260 was used.

	Double-and-Add		Addition Chains	
ALUs	4			2
Slices	3,693 (6%)			1,879 (3%)
min. period (ns)	24.681			22.437
Clock Freq.	40.52Mhz			44.57Mhz
PA Algorithm	2	5	2	5
Time (ms)	6.94	5.75	9.05	4.7

TABLE IV

POST PLACE AND ROUTE RESULTS FOR XILINX XC2VP125

As described in Section V-A.1, multiplication is executed in $p_b + 2$ clock cycles. Additions and subtractions take 2 clock cycles. For a field size of 160 and performing all multiplications and add/subs in series, point addition described by Algorithm 2 is executed in 2,444 cycles while a point doubling is executed in 1,636 cycles. Using the improved formulae from Algorithm 5 a point addition can be executed in 1,150 clock cycles, half that of the previous algorithm for point addition.

From the results in Table IV it can be seen that the new algorithm for point addition gives a slight improvement in execution time for the *double and add* method. Since, the addition chain method does not depend on point doubling, the effect of the new point addition algorithm is much greater. With the improved point addition using addition chains we get the best results over execution time and area.

Using addition chains, the dedicated modular adder, subtractor and Montgomery multiplier were also implemented in the processor. With a configuration of 1 adder, 1 subtractor and 2 Montgomery multipliers, the processor consumes 2,282 Slices (4% of the target device) and operates at 71.628Mhz. Although there is a slight increase in area consumption, the faster clock frequency means a point scalar multiplication for a 192-bit field size executes in 2.959ms.

VI. CONCLUSIONS

In this paper, a reconfigurable cryptographic processor has been used to efficiently compare two algorithms for elliptic curve point multiplication. Using addition chains we have found that not only can the cryptosystem resist SPA attacks but can also outperform a double-and-add approach. An improvement in execution time was also found when implementing dedicated units instead of the configurable ALU. This is at the expensive of a slight increase in overall area.

REFERENCES

[1] V. Miller, "Use of Elliptic Curves in Cryptography," *CRYPTO '85, Lecture Notes in Computer Science*, pp. 417–426, 1986.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Math. Computat.*, vol. 48, pp. 203–209, 1987.

[3] N. Koblitz, A. Menezes, and S. Vanstone, "The State of Elliptic Curve Cryptography," *Design Codes and Cryptography*, vol. 19, pp. 173–193, 2000.

[4] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *Lecture Notes in Computer Science*, vol. 1666, pp. 388–397, 1999.

[5] M. Nicolas, "Fast and secure elliptic curve scalar multiplication over prime fields using special addition chains." Cryptology ePrint Archive, Report 2006/216, 2006. http://eprint.iacr.org/.

[6] A.Daly, W. Marnane, and E. Popovici, "Fast Modular Inversion in the Montgomery Domain on Reconfigurable Logic," *ISSC*, vol. 19, pp. 363–367, 2003.

[7] S.B.Ors, E.Oswald, and B.Preneel, "Power-analysis attacks on an fpga - first experimental results," *Lecture Notes in Computer Science*, vol. 2279, pp. 35–50, 2003.

[8] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "An FPGA Implementation of a GF(p) ALU for Encryption Processors," *Elsevier Journal on Microprocessors and Microsystems (Special Issue on FPGAs:Applications and Designs)*, vol. 28, no. 5-6, pp. 253–260, 2005.

[9] P.L.Montgomery, "Modular multiplication without trial division," *Mathematics of Computations*, vol. 44, pp. 519–521, 1985.

[10] P. Leong and I. Leung, "A Microcoded Elliptic Curve Processor Using FPGA Technology," *IEEE Trans. on VLSI Systems*, vol. 10, no. 5, pp. 550–559, 2002.

[11] F. Crowe, A. Daly, and W. Marnane, "A Scalable Dual Mode Arithmetic Unit for Public Key Cryptosystems," *ITCC*, vol. 1, pp. 568 – 573, 2005.

[12] A. Byrne and W.P.Marnane, "Versatile Processor For GF(p^m) Arithmetic for use in Cryptographic Applications," *to appear in 24th IEEE Norchip Conference*, 2006.