Fully Parameterizable Elliptic Curve Cryptography Processor over GF(2^m)

Tim Kerins[1], Emanuel Popovici[1], William Marnane[1], and Patrick Fitzpatrick[2]

[1] Dept of Electrical and Electronic Engineering,
University College Cork, College Rd., Cork City, Ireland.
{timk,emanuel,liam}@rennes.ucc.ie
[2] Dept of Mathematics,
University College Cork, College Rd., Cork City, Ireland.
p.fitzpatrick@ucc.ie

Abstract. In this paper we present an Elliptic Curve Point Multiplication processor over base fields GF(2^m), suitable for use in a wide range of commercial cryptography applications. Our design operates in a polynomial basis is fully parameterizable in the irreducible polynomial and the chosen Elliptic Curve over any base Galois Field up to a given size. High performance is achieved by use of a dedicated Galois Field arithmetic coprocessor implemented on FPGA. The underlying FPGA architecture is used to increase calculation performance, taking advantage of the properties of this kind of programmable logic device to perform the large number of logical operations required. We discuss the performance of our processor for different Elliptic Curves and compare the results with recent implementations in terms of speed and security.

1 Introduction

In recent years a number of forces have elevated the importance of encryption. The expansive growth of the Internet has greatly increased the degree to which electronic systems connect and communicate. Public Key Cryptography is the basis of secure communication between unfamiliar hosts and is vital for secure E-commerce transactions. All standard mechanisms of network security such as Secure IP, Secure Sockets Layer (SSL) and Wireless Encryption Protocol (WEP) involve some form of public key cryptography. With bandwidth at a premium on E-commerce servers is it vital to have as much security per key bit as possible. Public key Elliptic Curve Cryptosystems (ECC) were introduced independently by Miller [1] and Koblitz [2]. The main strength of ECC is that the underlying mathematical problem is orders of magnitude more difficult than conventional public key algorithms so as a result the underlying key size can be made much smaller while retaining an equivalent level of security. For example a keysize of 200 bits in an ECC offers equavalent security to 1400 bit RSA [3]. Although elliptic curve systems offer greater security and utilize less bandwidth than conventional public key protocols the underlying arithmetic cannot be calculated as

efficiently leading to slower communication. In contrast to RSA no single standard has been set for ECC and a large number of curves are currently in use, thus an ECC processor must be able to handle different curves and underlying fields.

The speed of ECC is dependent on the speed at which the underlying arithmetic can be calculated. A recent approach has been to use dedicated hardware to speed up these operations [4]. Many hardware implementations of processors have concentrated on curves over a particular field or particular class of fields and involve a large amount of precomputation and reconfiguration. This approach is impractical in the case of electronic transactions where it may be necessary to quickly vary the encryption strength/speed ratio where the parameters of the system change frequently. As an example consider a SSL server used for electronic transaction. Perhaps some communication is required at 151 bit encryption but needs to be performed at a very high speed. At a later time 239 bit encryption is required where speed is less critical. Our approach is to implement a dedicated arithmetic coprocessor on FPGA as part of a fully parameterizable ECC system. In contrast to other recent implementations it can operate on any required key size, up to a maximum size, without reconfiguration. This addresses the problem of non standardization as the ECC can be adjusted to easily operate on any elliptic curve on a per application basis. It is envisioned that this design could be used as part of a larger library of FPGA cryptography ciphers as suggested in [5]. FPGAs are very suitable for this type of application as they allow for rapid changes in protocol without any changes in underlying hardware and are well suited for applications relating to internet security [6].

2 Related Work

A number of implementations have been documented for the computation of point scalar multiplication, which is the basic operation used by ECC. Processors based on composite fields GF($(2^m)^n$) are reported in [7] and although they are efficient, questions have been raised about the mathematical security of this type of approach [8]. High performance FPGA and software implementations of ECC systems are reported in [9]. A significant stand alone elliptic curve processor is documented in [10]. In this case parameterization is achieved by reconfiguration of the FPGA. This adds an extra level of complexity to the design as reconfiguration time must be taken into account when switching field size. Our work is similar to [11] except that we achieve rapid parameterization of an ECC without reconfiguration. We believe our processor is an improvement over its predecessors due to its simplicity of design and efficient use of underlying hardware to achieve full parameterization. Our Galois Field processor design is based on that presented in [12].

3 Elliptic Curve Cryptography

To encrypt data in ECC, it is represented as a point on an elliptic curve over a finite field. In common with other public key schemes it relies on the existence of a "trapdoor function", ie one which is easy to perform but very difficult to reverse. In this case, when a point is added to itself a number of times, it is computationally intractable to recover the original point without knowing the exact number of additions. This is where the security of ECC resides. In some sense the number of point additions represents the secret key while a point on the curve represents the data to be encrypted. In this section we give a brief overview of the mathematical basis of elliptic curve cryptography. Additional information on elliptic curves and their applications is given in [3].

3.1 Elliptic Curve Arithmetic

The fundamental operation for ECC is point scalar multiplication, i.e. a point is added to itself k times.

$$\begin{aligned} Q &= kP \\ &= \underbrace{P \oplus P \oplus ... \oplus P}_{k\ times} \end{aligned} \tag{1}$$

Point addition is defined geometrically by the "chord-tangent" law of composition $\oplus$.

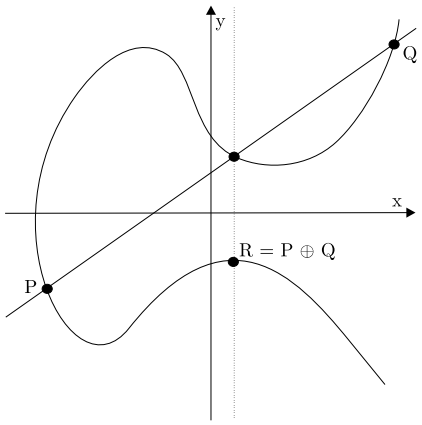

Fig. 1. Adding two points on an Elliptic Curve

To add two distinct points P and Q on an elliptic curve a chord is drawn between then and the third point of intersection of this line with the curve is reflected through the x axis. The reflection $R = P \oplus Q$. Adding a point P to itself (doubling P) is performed in a similar manner. In this case the tangent

to the curve at P is taken. This line intersects the curve at exactly one other point which is then reflected through the x axis. This reflection is $2P = P \oplus P$. Point scalar multiplication is performed by successive doublings and addings of the base point P.

3.2 Elliptic Curves over Binary Finite Fields

For our implementation we choose elliptic curves over the field GF(2^m). The advantage of such a representation is that field elements are represented in hardware as m bit registers and addition of elements is a bitwise XOR. Field elements are represented in the canonical (polynomial) basis and multiplications and divisions are performed modulo a suitable irreducible polynomial p. A representation of an elliptic curve E, in affine coordinates, suitable for cryptography is defined to be the set of solutions $(x, y) \in GF(2^m) \times GF(2^m)$ to the equation

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

where $a, b \in GF(2^m), b \neq 0$. Explicit rational formulas for the addition rule involve several operations in the underlying finite field. Given that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are points on the elliptic curve then $R = (x_3, y_3) = P \oplus Q$ is defined when $P \neq Q$ (point addition) as

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \tag{3}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \tag{4}$$

$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1 \tag{5}$$

and when $P = Q$ (point doubling) as

$$\lambda = x_1 + \frac{y_1}{x_1} \tag{6}$$

$$x_3 = \lambda^2 + \lambda + a \tag{7}$$

$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1 \tag{8}$$

In either case the operation involves 1 division, 1 multiplication and 1 squaring in the underlying finite field. Traditionally inversion over a Galois Field is an expensive operation in terms of clock cycles and hardware. The speed at which it can be performed typically limits the speed at which an ECC over a polynomial basis in affine coordinates can operate.

3.3 Elliptic Curve Cryptosystems

Elliptic curve cryptography is based on the discrete logarithm problem applied to elliptic curves over a finite field. In particular for an elliptic curve E it relies on the fact that it is relatively easy to compute equation 1 while there is currently no

known subexponential time algorithm to compute k given P and Q. In practice P is chosen so it is a generator of a finite cyclic subgroup of points on the curve and k is less than the order of this group. In implementing an elliptic curve cryptosystem a number of choices have to be made each of which will affect security and performance

- Size of underlying finite field GF(2^m).
- Irreducible polynomial for finite field arithmetic $p(x)$.
- Suitable elliptic Curve E.
- Suitable base point P.

In a modern commercial communication environment an ECC processor needs to be parameterizable in each of the above so that the speed and strength of the cipher can be easily varied. There are elliptic curve analogs of most popular public key protocols such as elliptic curve Diffie-Hellman (ECDH) and elliptic curve ElGamal [3]. They are also used for secure hash function generation and digital signatures [3]. For example in ECDH a random base point P is chosen, this represents the public key, while k represents the private key. The shared secret key S between two parties A and B is easily calculated by

$$S = k_A(k_B P) = k_B(k_A P) \tag{9}$$

4 An Elliptic Curve Processor

Our processor uses FPGA technology to perform the most time critical operations, namely the Galois Field arithmetic, and its design is well suited to this type of application. It has a highly efficient inversion routine and ability to perform three field operations in parallel. The processor involves a dedicated Galois Field arithmetic coprocessor on FPGA, realized on the Celoxica PC-1000 bus "plug-in" card [13], containing a VirtexE2000 device [14]. The underlying Galois Field operations involved in the point addition (equations 3,4,5) and point doubling (eqations 6,7,8) are performed on FPGA, in communication with a host PC (Dell pentium 4 1.8 Ghz processor), which performs the scalar multiplication by successive point doublings and addings.

4.1 The ABC Galois Field Coprocessor

Our Galois Field processor design was first proposed in [12]. It performs the calculation

$$u = \frac{ab}{c} \ mod \ p \tag{10}$$

in $2m$ clock cycles, where $u, a, b, c \in GF(2^m)$ and p is the irreducible polynomial. An outline of the algorithm is displayed in figure 2. The (u_i, v_i) i=1,2,3 pairs are pairs of polynomials of maximum order m (m bit registers) and the discrepancies d_1 and d_2 are the x^ith coefficients (i^{th} bit in register) in

$$\begin{matrix} u_1 c + v_1 p \\ u_2 c + v_2 p \end{matrix} \tag{11}$$

respectively. The discrepancy d_3 is given by the x^ith coefficient in

$$u_3 c + v_3 p + ab \tag{12}$$

These operations are performed in the function ComputeDiscrepancies in figure 2 and are explained in detail in the next subsection. The pair (u_1, v_1) is defined to be smaller than (u_2, v_2) if the degree of u_1 is less than or equal to the degree of $u_2 + 1$. Let $k1$ denotes the index of (u_1, u_2) and $k2$ denotes the index of (u_2, v_2).

```
ABCprocessor(a,b,c,p){              // computes u=(ab/c)mod p
  var
    d1=0, d2=1, d3=0;               // initialize discrepancies
    u1=1, v1=0;                     // initialize (u1,v1) registers
    u2=0, v2=1;                     // initialize (u2,v2) registers
    u3=0, v3=0;                     // initialize (u3,v3) registers
    k1=0, k2=1;                     // order of (u1,u2), (v1,v2)
    i;                              // iteration counter
  begin
    for(i=0; i<2*m; i++){
      ComputeDiscrepancies(a,b,c,p,d1,d2,d3);
        if((k1<=k2 && d1=1 && d2=1) || (d2=0 && d1=1)){
          u2=u2 XOR (u1 AND d2);    // XORs u2,u1 registers if d2=1
          v2=v2 XOR (v1 AND d2);    // XORs v2,v1 registers if d2=1
          u3=u3 XOR (u1 AND d3);    // XORs u3,u1 registers if d3=1
          v3=v3 XOR (v1 AND d3);    // XORs v3,v1 registers if d3=1
          u1=u1<<1;                 // left shift u1 : multiply by x
          v1=v1<<1;                 // left shift v1 : multiply by x
          k1=k1+1;}                 // increment order (u1,v1)
        else{
          u1=u2 XOR (u1 AND d1);    // XORs u2,u1 if d1=1
          v1=v2 XOR (v1 AND d1);    // XORs v2,v1 if d1=1
          u3=u3 XOR (u2 AND d3);    // XORs u3,u2 if d3=1
          v3=v3 XOR (u3 AND d3);    // XORs v3,u3 if d3=1
          u2=u2<<1;                 // left shift u2 : multiply by x
          v2=v2<<1;                 // left shift v2 : multiply by x
          k2=k2+1;}                 // increment order (u2,v2)
    return(u3);}                    // return result
```

Fig. 2. Pseudocode for the ABC Coprocessor

To perform arithmetic on a smaller field, the irreducible p' that generates the smaller Galois Field is loaded to the SRAMS from the PC and the same hardware is used for calculation of equation 10 for these smaller field elements, with faster calculation time depending on the degree of p'.

4.2 ABC Coprocessor Architecture on FPGA

Our processor used thirteen, 256 bit registers to calculate equation 10. Three holding registers are used for the initial loading of data, and the other ten are used for calculation as illustrated in figure 3. Initially the holding registers and register b are filled. During calculation the registers a, c and p are loaded bit serially from the holding registers and the (u_i,v_i) registers shifted or XORed as illustrated in figure 2. The calculation of the discrepancies (d_1,d_2 and d_3) is achieved by XORing the results of the Sum Blocks (Σ) as illustrated in equation 11 and equation 12. The Sum Blocks are the most hardware intensive part of the calculation and are well suited to implementation on FPGA as they can exploit the underlying technology. The architecture of a Sum Block for two 8 bit registers X and Y is illustrated in figure 4.

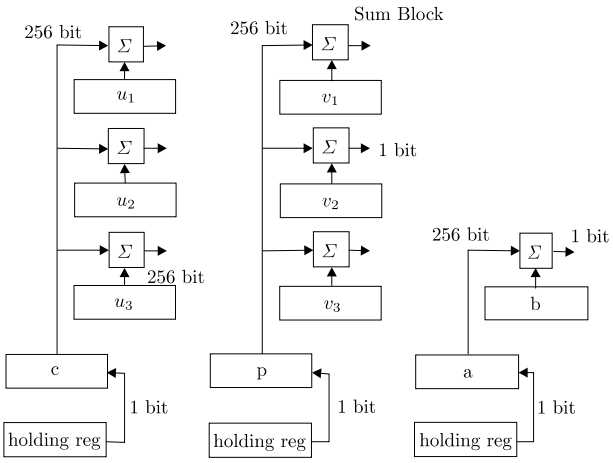

Fig. 3. The ABC coprocessor architecture

In our case with 256 bit registers each Sum Block has 8 levels of logic and represents the longest path delay in the ABC design. Our design contains 7 Sum Blocks.During synthesis one 256 bit Sum Bock is mapped to 68 CLBs. By mapping the logic to both the 4 bit LUTs and the carry XOR gates a sum block is reduced in complexity to only 3 levels of logic and optimizes CLB usage on the device as well as increasing overall performance.

4.3 Eliptic Curve Cryptosystem using ABC Coprocessor

A schematic of our ECC processor is illustrated in figure 5. The ABC processor was developed in a VHDL environment using the Synopsys FPGA compiler tool and Xilinx Design Manager. The point scalar multiplication operation along with the maping of data to the SRAMS were coded in the C programming language,

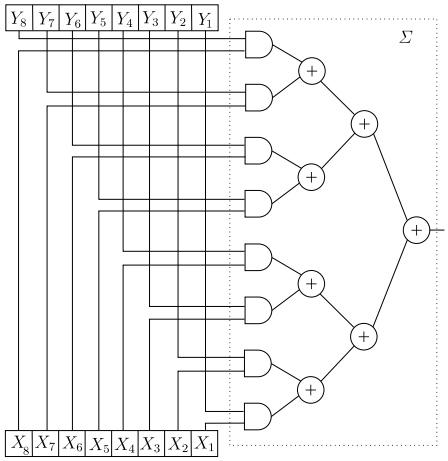

Fig. 4. The Σ Sum Block for 8 bit registers X and Y

with the underlying Galois Field arithmetic being performed on the FPGA. Control, and the order of the field are communicated directly to the FPGA. Data is read from the PC in 32 bit blocks into the RC-1000 SRAMS by direct memory mapping. The FPGA then has control of the SRAMS and reads each of p, a, b, c in 128 bit blocks. After calculation the result $u3$ is written back to the SRAMS, overwriting data c. The PC then has control of the SRAMS and accesses the result.

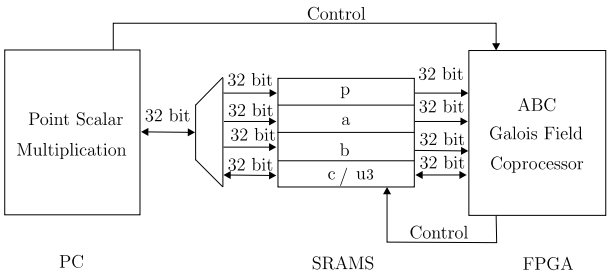

Fig. 5. Elliptic Curve Cryptography Processor

The required Galois Field arithmetic is performed by inputting data from equation 3 - 8 to the processor. For example to perform equation 6, $a := y_1$, $c := x_1$, $b := i$ (where i is the identity element 00...001), and the XOR operation is performed at the PC. In each point addition/doubling the ABC coprocessor is used 3 times. Using equation 1 the number of point doubling operations is approximately m (where m is the order of the field extension) and the number of point additions is approximately $m/2$ (given that the hamming weight of the

binary expansion of k is $0.5m$). Thus to perform an elliptic curve point scalar multiplication the ABC processor is accessed approximately $4.5m$ times.

5 Results

The ABC processor was implemented on the Xilinx chip VX2000Bg560-6. For a 256 bit register size (maximum field GF(2^{255}) it was found to occupy 4048 slices, giving an equivalent gate count of 74103. This occupies approximately 20% of the total device. Using Xilinx Design Manager a clock frequency of 40 MHz was achieved for this design. As the access time between the SRAMS and PC is neglible at our data rates [13] and as the PC and FPGA can operate in parallel, an elliptic curve point scalar multiplication is calculated in $9m^2$ clock cycles. Execution times and data rates for various fields are illustrated in table 1.

Table 1. Execution Times and Data Rates for elliptic curve point scalar multiplication

Field Size (bits)	Time (ms)	Data Rate (Kbits/sec)
151	5.1	29.6
176	6.9	25.5
191	8.2	23.3
239	12.8	18.6

Our design performs well in comparison with [11], reporting a similar or decreased calculation time for point scalar multiplication, along with the added advantage that full parameterization is achieved without reconfiguration, for curves over any field up to GF(2^{255}). Reconfiguration time for Virtex Devices is known to be of the order of milliseconds unless dynamic reconfiguration is used [15]. The Galois Field coprocessor circuitry does not depend on the operation it is performing so it should be fully resistant to power analysis attacks. This in conjunction with a suitable method of securing FPGA to PC communications against side channel attacks increases the overall security of the system. By reconfiguration of the FPGA our processor can be expanded to GF(2^{1023}) with a degradation in performance.

6 Conclusions and Further Work

We have presented an implementation of an ECC processor fully parameterizable in the order of the underlying binary field m, the irreducible polynomial used to generate the field $p(x)$ and the elliptic curve used E. FPGA technology was used to perform the most time critical operations and performance compares well with recent implementations. It is well suited to applications where rapid parameterization is required (e.g. SSL servers) and can be included in a

larger FPGA cipher library. As operations are carried out in the polynomial basis, basis conversions are not required when interfacing with other systems. We are currently investigating the effects of custom mapping the design to FPGA and moving more of the control logic to hardware. It is expected that these modifications will lead to large improvements in performance.

7 Acknowledgement

This work was carried out as part of an Enterprise Ireland Research Innovation Fund Project.

References

1. V. S. Miller: Use of Elliptic Curves in Cryptography. Advances in Cryptology Crypto'85, Lecture Notes in Computer Science No. 218, Springer-Verlag Berlin, (1985) 417-426.
2. N. Koblitz: Elliptic Curve Cryptosystems. Math Comp Vol. 48, (1987) 203-209.
3. I. Blake, G. Seroussi, N. Smart: Elliptic Curves in Cryptography, London Mathematical Society Lecture Note Series 265, Cambridge University Press (2000).
4. G. B. Agnew, R. C. Mullin, S. A. Vanstone: An implementation of elliptic curve cryptosystems over $F_{2^{155}}$, IEEE Transactions on Selected Areas in Communications. Vol. 11, (1993) 804-813.
5. T. Kerins, E. Popovici, A. Daly, W. Marnane: Hardware Encryption Engines for E-Commerce, Proceesings of Irish Signals and Systems Conference (2002) 89-94.
6. A. Dandalis, V. K. Prasanna: FPGA based cryptography for Internet Security, Online Syposium for Electronic Engineers, Nov 2000.
7. M. Rosner: Elliptic Curve Cryptosystems on Reconfigurable Hardware, Masters Thesis, ECE Dept., Worchester Polytechnic Institute, Worchester USA (1998).
8. P. Gaudry, F. Hess, N. P. Smart: Constructive and Destructive Facets of Weil Descent on Elliptic Curves, Journal of Cryptology, Vol. 15, No.1 (2002) 19-46.
9. D. Hankerson J. L. Hernadez A. Menezes: Software Implementation of Elliptic Curve Cryptography Over Binary fields, Proceeeding of CHES 2000, Lecture Notes in Computer Science, 1965 Springer-Verlag, (2000) 1-24.
10. G. Orlando, C. Paar: A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m), Proceedings of CHES 2000, Lecture Notes in Computer Science 1965, Springer-Verlag (2000). 41-56
11. K. H. Leung, K. W. Ma K. W. Wong P. H. W. Leong: FPGA implementation of a Microcoded Elliptic Curve Cryptographic Processor: Proceedings of Field Field-Programmable Custom Computing Machines (2000). 68-76
12. E. M. Popovici, P. Fitzpatrick: Algorithm and Architecture for a Galois Field Multiplicative Arithmetic Processor, submitted IEEE Trans on Information Theory Jan 2002.
13. http://www.celoxica.com.
14. http://www.xilinx.com.
15. A. G. Popovici: Reconfigurable Computing, Masters Thesis, National Microelectronics Research Centre, Cork Ireland. (2001).