Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2^{163}) for ECC protocols

YongJe Choi[1], HoWon Kim[1] and MooSeop Kim[1]
[1]Electronics and Telecommunications Research Institute
161 KaJeongDong YuSeongGu, DaeJeon, 305-350, KOREA
Tel: +82-42-860-1327, FAX: +82-42-860-5611
E-mail: choiyj@etri.re.kr

Abstract: This paper describes the design of elliptic curve crypto (ECC) coprocessor over binary fields for ECC protocols. Our ECC processor provides the elliptic curve operations for Diffie-Hellman, EC Elgamal and ECDSA protocols. The ECC we have implemented is defined over the field GF(2^{163}),which is a SEC-2 recommendation [6].

1. Introduction

The Elliptic Curve Cryptography (ECC) was proposed in 1985 by Neal Koblitz[3] and Victor Miller[4], and the security of it rests on the discrete logarithm problem over the points on an elliptic curve. The ECC provides higher strength-per-bit than any other current public-key cryptosystems [1]. Because of its higher strength-per-bit, Elliptic Curve Cryptosystems are being increasingly used in practical applications (e.g. IC card and mobile devices) instead of RSA, which is most used for public-key cryptosystems.

The Elliptic Curve Cryptosystems are used for implementing protocols such as ECDSA digital signature scheme, EC Elgamal Encryption/Decryption scheme, Diffie-Hellman key exchange scheme and so on. This paper analyzed the elliptic curve operations of these ECC protocols and designed the Elliptic Curve Cryptographic Coprocessor to efficiently implement the ECC system.

2. ECC arithmetic

2.1 Scalar Multiplication

Just as modular exponentiation determines the efficiency of RSA cryptographic systems, scalar multiplication dominates the execution time of ECC systems. Scalar multiplication is the operation to compute kP, where k is a random integer and P is an elliptic curve point, and it can be defined the combination of additions of two points on an elliptic curve. The addition of two points on an elliptic curve is defined in order that the addition results will be another points on the curve as following Algorithm 1. (The elliptic curve over F_{2^m} given by the equation $y^2 + xy = x^3 + ax^2 + b$ and P_1 and P_2 are on the elliptic curve).

Algorithm 1. Point Addition Equation

Input : $P_1 = (x_1, y_1)$, $P_2 = (x_2 , y_2)$.
Output : $P_3 = P_1 + P_2 = (x_3 , y_3)$.

1. If $P_1 = P_2$ (doubling)
   $x_3 = \lambda^2 + \lambda + a$, $y_3 = x_1^2 + (\lambda + 1) x_3$
   where $(\lambda = x_1 + y_1 / x_1)$
2. Else if $P_1 \neq P_2$ (point addition)
   $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$,
   $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$
   where $(\lambda = (y_2 + y_1) / (x_2 + x_1))$
3. Return (x_3 , y_3)

In either case, when $P_1 = P_2$ (doubling) and $P_1 \neq P_2$ (point addition), major operations are field multiplication and field inversion. (Squaring and field addition are enough ignorable because of its less computation time.)

2.2 Field Multiplication

The shift-and-add method for field multiplication is well suited for hardware implementation because a vector shift can be performed in one clock cycle, while the large number of word shifts make it less desirable for software implementation, and the modular reduction can be operated simultaneously. This algorithm is based on the observation that $a \bullet b \bmod f(x) = (a_{m-1} x^{m-1} b + \bullet\bullet\bullet + a_2 x^2 b + a_1 x b + a_0 b) \bmod f(x) = (a_{m-1} x^{m-1} b \bmod f(x)) + \bullet\bullet\bullet + (a_2 x^2 b \bmod f(x)) + (a_1 x b \bmod f(x)) + (a_0 b \bmod f(x))$ as following Algorithm 2.In step 2, b is added to the accumulator c if $a_i = 1$ and shifts the result to the left $(c \bullet x)$. 'Mod $f(x)$' operation can be easily computed by the addition of $f(x)$ to b if $c_m = 1$.

Algorithm 2. Shift-and-add field multiplication

Input : Binary Polynomials $a(x)$ and $b(x)$.
Output : $c(x) = a(x) \bullet b(x) \bmod f(x)$.

1. $c \leftarrow 0$.
2. For i from $m-1$ to 1 do
   2.1 If $a_i = 1$ then $c \leftarrow c + b$.
   2.2 $c \leftarrow c \bullet x \bmod f(x)$.
3. If $a_0 = 1$ then $c \leftarrow c + b$.
4. Return (c).

2.2 Field Inverse Multiplication

Almost Inverse Algorithm (AIA) is desirable for hardware implementation because it is calculated by one bit (left or right) shift operations and bit-width XOR operations [7]. The reduction step of AIA can be performed whenever u is divided by x. This method, it is

called Modified Almost Inverse Algorithm (MAIA)[2], reduces hardware sizes and operating cycles. In algorithm 3, note that if b is not divisible by x, then b is replaced by $b + f$ (and d by $d - 1$) before the division. When operation step is terminated, $u = 1$ and $b = a^{-1} \bmod f(x)$.

Algorithm 3. Modified Almost Inverse Algorithm

Input : $a \in F_2^m$, $a \neq 0$.
Output : $a^{-1} \bmod f(x)$.

1. $b \leftarrow 1, c \leftarrow 0, u \leftarrow a, v \leftarrow f, k \leftarrow 0$.
2. While x divides u do :
   2.1 $u \leftarrow u / x$.
   2.2 If x divides b then $b \leftarrow b / x$;
   else $b \leftarrow (b + f) / x$.
3. If $u = 1$ the return (b).
4. If $\deg(u) < \deg(v)$ then : $u \leftrightarrow v$, $b \leftrightarrow c$.
5. $u \leftarrow u + v, b \leftarrow b + c$.
6. Goto step 2.

3. ECC Protocols

3.1 Diffe-Hellman

The Diffie-Hellman protocol is the basic public key crypto system proposed for secret key sharing. A (Alice) and B (Bob) first agree to use a specific curve, field size, and type of mathematics. They then share the secret key by process as follows. We can see that we just need scalar multiplication in order to implement the Diffie-Hellman protocol.

Algorithm 4. Diffie-Hellman Protocol

1. A and B each chose random private key 'k_a' and 'k_b'.
2. A and B each calculate k_aP and k_bP, and send them to opposite side.
3. A and B both compute the shared secret Q $= k_a(k_bP) = k_b(k_aP)$.

3.2 EC Elgamal

Elliptic Curve (EC) Elgamal protocol is used for secret message translation between A(Alice) and B(Bob). Message is encrypted by A's public key, and decrypted by A's secret key. Algorithm 5 presents the EC Elgamal protocol for translating Message (M_1, M_2) to A.

Algorithm 5. EC Elgamal Protocol

Key generation : (A)

1. Select a random integer a from [1, n-1].
2. Compute aP.
3. A's public key is aP; A's private key is a.

Encryption : (B)

1. Select a random integer k from [1, n-1].
2. Compute kP and $k(a\text{P}) = a(k\text{P}) = (x,y)$.
3. If $x = 0 \pmod p$ or $y = 0 \pmod p$ then go to step 2.
4. Compute $M_1 * x$ and $M_2 * y$.
5. Send (kP, M_1 x, M_2 y) to A.

Decryption : (A)

1. Compute $a(k\text{P}) = (x, y)$.
2. Compute M_1 x / x and M_2 y / y, then recover the message $M = (M_1, M_2)$.

As the algorithm 5, the EC Elgamal protocol is implemented by scalar multiplication, field multiplication and field inverse multiplication.

3.3 ECDSA

EC Digital Signature Algorithm (DSA) is the elliptic curve analogue of the DSA, which is most famous signature protocol. This protocol needs not only the elliptic curve operations, such as scalar multiplication, field multiplication and field inverse multiplication, but also big integer multiplication, big integer inverse multiplication, modular operation and SHA-1, which is the 160-bit hash function. In the ECDSA, A (Alice) generates the signature with his secret key and B (Bob) verifies the signature with A's public key. Algorithm 6 is the ECDSA protocol which A signs the message m, and B verifies A's signature.

Algorithm 6. ECDSA Protocol

Key generation : (A)

1. Select a random integer d from [1, n-1].
2. Compute Q = dP.
3. A's public key is Q; A's private key is d.

Signature generation : (A)

1. Select a random integer k from [1, n-1].
2. Compute kG = (x_1, y_1) and $r = x_1 \pmod n$.
3. If r = 0 then go to step 1.
4. Compute $k^{-1} \pmod n$.
5. Compute $s = k^{-1} \{\text{SHA-1}(m) + dr\} \pmod n$.
6. If s = 0 then go to step 1.
7. Send m and (r, s), which is A's signature for the message m, to B.

Signature verification : (B)

1. Verify that r and s are integers in [1, n-1].
2. Compute e = SHA-1(m).
3. Compute $w = s^{-1} \pmod n$.
4. Compute $u_1 = e * w \pmod n$ and $u_2 = r * w \pmod n$.
5. Compute $u_1P + u_2Q = (x_1, y_1)$ and $v = x_1 \pmod n$.
6. Accept the signature if and only if v = r.

As can be seen in the above algorithm, we need various operations for implementing the ECDSA protocol. But we are only concerned with the elliptic curve operations, because these operations dominate the execution time of the protocol schemes. In this protocol, there are two operations, which are the scalar multiplication in signature verification step 5 and signature generation step 2 and the point addition in signature verification step 5.

4. ECC coprocessor implementation

4.1 ECC coprocessor structure

Figure 1 shows a structure of our ECC coprocessor. The ECC coprocessor consists of an interface block, a register file block, an ECC adder, a degree comparator block and an ECC controller. The ECC coprocessor we have implemented is defined over the field GF (2^{163}), which is a SEC-2 recommendation [6], with this field being defined by the field polynomial $F(x) = x^{163} + x^7 + x^6 + x^3 + 1$.

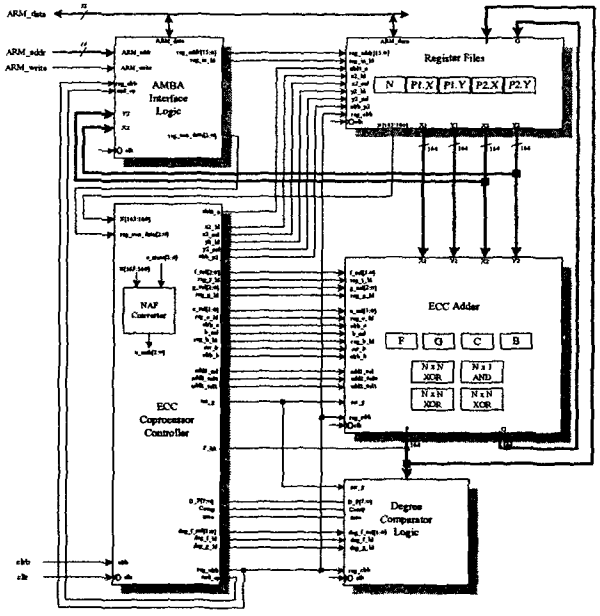

Figure 1. Block diagram of the ECC coprocessor

The interface block controls the communications between main processor and coprocessor. It includes the Advanced Microcontroller Bus Architecture (AMBA) for the testing on the ARM emulator system. The register file block stores the input, output, and intermediate values. It has five registers, which are the N register, the P1.x register, the P1.y register, the P2.x register, and the P2.y register. One polynomial multiplier and one polynomial inverse multiplier and two polynomial adders are merged into the ECC adder block so as to perform the scalar multiplication efficiently and optimize the hardware size. The polynomial multiplication and the polynomial inverse multiplication can be also calculated independently. At chapter 2, these two operations are need for the EC Elgamal protocol implementation. Figure 2 shows the block diagram of the ECC adder block.

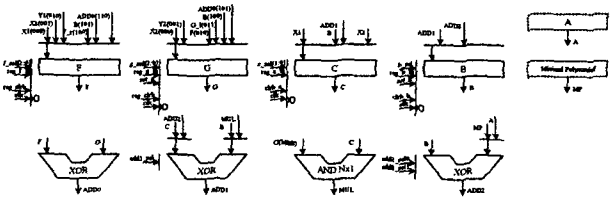

Figure 2. Block diagram of the ECC adder block

The degree comparator block, show in figure 3, is used for finding and comparing degree of polynomials when the polynomial inverse multiplication is performed at the ECC adder. It also determines whether the input points of the ECC adder are the infinite points or not.

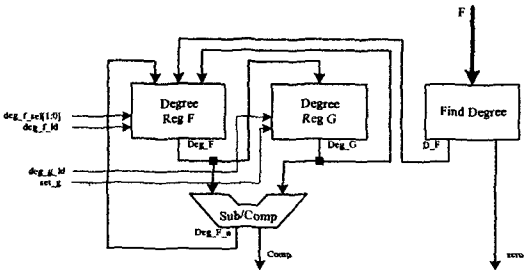

Figure 3. Block diagram of the degree comparator block

The ECC controller controls all other blocks according to the value of control register. It has built-in the Non-Adjacent Format (NAF) converter for enhancing the performance of the ECC coprocessor; NAF method for scalar multiplication can reduce the point addition operations by approximately 17%.

4.2 ECC coprocessor operations

Our ECC coprocessor can do four independent operations for ECC protocols implementation, which are a scalar multiplication, an Elliptic Curve point addition, a polynomial (field) multiplication and a polynomial (field) inverse multiplication. These operations are performed differently according to controller' s behavior and the input-output registers' action in the register file block. Table 1 presents the control register and the operations of

the ECC coprocessor.

Table 1 : ECC coprocessor operations

Control Register Bit	ECC coprocessor operation	
	Input	Output
0	Coprocessor start/stop	
1	Scalar multiplication	
	N ← k P1.x, P1.y ← P(x, y)	P2.x, P2.y ← kP
2	EC point addition	
	P1.x, P1.y ← P(x, y) P2.x, P2.y ← G(x', y')	P2.x, P2.y ← P + G
3	Polynomial multiplication	
	P1.x ← a(x) P2.x ← b(x)	P2.x ← a(x) * b(x)
4	Polynomial inverse multiplication	
	P1.x ← b(x) P2.x ← a(x)	P2.x ← a(x) / b(x)
5	Coprocessor internal register reset	

5. ECC coprocessor performance

We have measured the performance of our ECC processor with an ARM emulator board, shown in Figure 4. The crypto processor is implemented on the Xillinx Virtex-1000 FPGA of the emulator board and tested at 20MHz clock frequency. The results are shown in Table 2. It is also synthesized with SAMSUNG 0.35μm CMOS technology and has a hardware size of 18k gates.

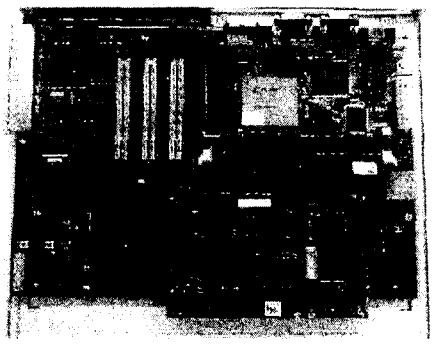

Figure 4. Photograph of the ARM emulator board for testing the ECC processor

Table 2. ECC processor operation times

Operation	Operation time
Scalar multiplication (kP)	12.3 ms
Point addition (P + G)	56 μs
Polynomial multiplication a(x) * b(x)	9 μs
Polynomial inverse multiplication (a(x) / b(x))	37 μs

6. Conclusion

We have analyzed the ECC protocols and designed the ECC coprocessor over the field GF (2^{163}). The ECC processor can calculate various operations for implementing ECC protocols, which are a scalar multiplication, an Elliptic Curve point addition, a polynomial multiplication and a polynomial inverse multiplication. It is synthesized and tested with Xilinx FPGA and its average operation time for scalar multiplication is 12.3msec. For future works, we will include the high performance ECC processor, which is able to operate for various elliptic curve, and countermeasure hardware to prevent illegal attacks such as simple power analysis (SPA) attack and differential power analysis (DPA) attack.

References

[1] Certicom research, "*The Elliptic Curve Cryptosystem*", Certicom, April 1997.

[2] Darrel Hankerson, Julio Lopez Hernandez, Alfred Menezes, "*Software Implementation of Elliptic Curve Cryptography over Binary Fields*", CHES 2000, page 1-24. 2000.

[3] N. Koblitz, "*Elliptic curve cryptosystems*", Mathematics of Computation, number 48, pages 203-209, 1987.

[4] V.S. Miller, "*Use of elliptic curve in cryptography*", Advances in Cryptology – Proceedings of CRYPTO'85, Springer Verlag Lecture Notes in Computer Science 218, pages 417-426, 1986.

[5] R.L. Rivest, A. Shamir, and L.M. Adleman, "*A method for obtaining digital signatures and public-key cryptosystems*", Communications of the ACM, volume 21, pages 120-126, February 1978.

[6] Certicom research, "*SEC 2 : Recommended Elliptic Curve Domain Parameters*", October 1999.

[7] Richard Schroeppel, Hilarie Orman, Sean O'Malley, "*Fast Key Exchange with Elliptic Curve Systems*", TR-95-03(Tucson, AZ: University of Arizona, Computer Sciences Department, 1995)

[8] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, Handbook of Applied Cryptography, CRC press, 1997.