On the Hardware Design of an Elliptic Curve Cryptosystem

Miguel Morales-Sandoval and Claudia Feregrino-Uribe
National Institute for Astrophisics, Optics and Electronics
Computer Science Department
Luis Enrique Erro No. 1, Sta. Ma. Tonantzintla, Pue, 72840 Puebla, México
{mmorales, cferegrino}@inaoep.mx

Abstract

We present a hardware architecture for an Elliptic Curve Cryptography System performing the three basic cryptographic schemes: DH key generation, encryption and digital signature. The architecture is described by using hardware description languages, specifically Handel C and VHDL. Because of the sequential nature of the cryptographic algorithms, they are written in Handel C language. The critical part of the cryptosystem is a module performing the scalar multiplication operation. This module has been written in VHDL to let further improvements. The points of the elliptic curve are represented in projective coordinates working over the two-characteristic finite field and using polynomial basis. A prototype of this hardware architecture is implemented on a Xilinx Virtex II FPGA device.

1. Introduction

Because of the information processing and telecommunications revolutions, there is an increasing demand for techniques to keep information secret, to determine that information has not been forged and to determine who authored pieces of information.

Cryptographic techniques are currently being utilized for these purposes. Elliptic Curve Cryptography (ECC) [12] has been receiving a lot of attention in the last years because of the benefits it offers. ECC employs smaller length keys than other cryptosystems like RSA, what implies less space for key storage and less costly modular operations. Furthermore, it has been shown in the literature [12] that ECC's security is higher than that provided by RSA, which is the most widely used public key cryptosystem. Although ECC offers the same security level than RSA using smaller length key, among scientists and mathematicians still exists skepticism for using ECC in practical applications. ECC's security has not been proved; its strength is based on the inability to find attacks.

International organizations such as ISO, ANSI, IEEE and NIST have been working to standardize the use of ECC. Also, several enterprises like Certicom, Sun Microsystems, Motorola and others have been investing in research; these enterprises consider ECC as the cryptosystem of the future. The main area where ECC is applied is to implement cryptographic functions in constrained environments. Main applications of ECC are in the wireless market where security is required but devices have limited resources (memory and computational power) to implement any other public key cryptosystem. Since performance of all elliptic curve cryptosystems depends on the efficiency to perform field arithmetic operations, most of the reported papers are related to the improvement of such arithmetic units.

In this work, we present a hardware implementation of an elliptic curve cryptosystem. We have implemented the three basic cryptographic schemes: ECDH for key generation [4], ECIES scheme [19] to encrypt data and ECDSA [1] to generate a digital signature. Also, the SHA-1 algorithm [20] was implemented for authentication; in the ECDSA scheme it is necesary to get the hash value of the message to be signed and in the ECIES scheme it is required to generate a bit string which is used to encrypt the message. A hardware implementation is well suited since elliptic curve cryptography implies complex field and elliptic curve operations. To the best of our knowledge, a hardware architecture that implements the three cryptographic algorithms mentioned above has not been reported.

The most time consuming operation in an elliptic curve cryptosystem is the so-called scalar multiplication operation. In the DH key generation scheme it is necessary to perform one scalar multiplication operation; in the ECIES scheme, scalar multiplication is required twice in the encryption process and once in the decryption process; in the ECDSA scheme, this operation is required once in signature generation and twice in signature verification. We have implemented a coprocessor in VHDL for performing the scalar multiplication operation in the binary finite field $F_{2^{191}}$, using polynomial basis and projective coordinates to represent points in the elliptic curve. Elliptic curve cryptographic schemes mentioned above have been written in Handel C language including the SHA-1 [20] algorithm for authentication because of their more algorithmic approach. We have simulated the full system co-simulating the VHDL and Handel C code. Also, we have tested the blocks independently in a Xilinx Virtex II FPGA device.

The remainder of this paper is organized as follows: Section 2 shows the related work, Section 3 explains the fundamentals of ECC and Section 4 describes the algorithms implemented. In Section 5 the design of the cryptosystem is presented; the results obtained are summarized in Section 6. Finally Section 7 concludes this work.

2. Related Work

Since ECC was proposed in 1985 [11], [14], many results have been reported on secure and efficient implementations. The performance of the entire system depends mainly on the efficiency of the arithmetic units, which perform demanding finite field computations, and on fast algorithms for scalar multiplications. In addition, performance of ECC can be speeded up by selecting specific underlying finite fields and elliptic curves [2]. Both software and hardware ECC implementations have been reported, some of them are mentioned in the following subsections.

2.1. Software implementation

Some software implementations of ECC have been reported in [8], [21] and [13]. In [8] Hankerson *et al.* present a C implementation on a Pentium II workstation for performing the scalar multiplication. NIST random and Koblitz elliptic curves over the binary field were used. The best timing result for performing scalar multiplication was 1.68 ms for the binary field $F_{2^{163}}$ and 39.6 ms for $F_{2^{233}}$. In [21], an ECC implementation on a Palm OS Device is presented. A NIST (random and Koblitz) curve over the finite field $F_{2^{163}}$ was used and projective coordinates were selected. Code was written in C and executed on a Handspring Visor with 2 MB of memory. For the random curve, the scalar multiplication takes 3.5 sec using the Montgomery algorithm. For the Koblitz curve, the scalar multiplication takes 2.4 sec. 0.9 sec. In [13] an ANSI C software library for ECC is presented. The GNU Multiple Precision arithmetic library is used to perform the arithmetic operations over the a prime finite field. The library was tested on a Pentium III workstation using prime finite fields $F_{p^{175}}$, $F_{p^{192}}$ and $F_{p^{224}}$; for each of these finite fields, timing results for performing scalar multiplication were 13.6 ms, 15.7 ms and 19.5 ms respectively. In this last work, ECDH, ECES and ECDSA protocols were implemented to test the proposed library. For the finite field $F_{p^{192}}$, timing results are as follows: for ECES encryption it takes 36.5 ms for processing, for ECDSA digital signature generation it takes 22.7 ms long and 28.3 ms for verification.

Because of elliptic curve cryptographic algorithms perform a high amount of mathematical operations with large numbers in a finite field; implementing ECC with the available instruction set of a general purpose processor is inefficient, as mentioned in the software implementations. For this reason, a hardware solution could be better suited, especially for real time data processing.

2.2. Hardware implementations

Hardware architectures for ECC can be divided into processor or coprocessor approaches. In the former, there exist a number of specialized instructions the processor can support, some of them are for finite operations or elliptic curve points. In the latter, there are no such instructions because the algorithms are implemented directly on specialized hardware.

Some processors reported for ECC are found in [17], [18], [5] and [10]. In [17], Orlando and Paar have reported the fastest FPGA based processor while in [18] Satoh and Takano have reported the fastest ASIC based processor. The work performed by Orlando and Paar is considered as a benchmark, where the scalar multiplication can be performed in 0.21 ms, working over the binary field $F_{2^{167}}$ with polynomial basis representation and projective coordinates. The architecture exploits the benefits of reconfigurable computing and incorporates pipelining and concurrence techniques in order to have an optimized hardware that can support several elliptic curves and finite field orders. A prototype of this processor was implemented in a XCV400E-8 BG432 Xilinx FPGA. The processor reported by Satoh and Takano in [18] can support both prime and binary finite field for arbitrary prime numbers and irreducible polynomials. This characteristic is achieved by introducing a dual field multiplier. Scalar multiplication can be performed in 0.19 ms in the binary field $F_{2^{160}}$.

Some coprocessors reported for ECC are found in [16], [6] and [7]. In [16] Okada *et al.* reported a coprocessor for ECC both in a FPGA device and in an ASIC platform. In the case of the FPGA implementation, scalar multiplication takes 80 ms for random curve over the finite field $F_{2^{163}}$. The critical part of the coprocessor is a bit-parallel multiplier that can operate with different irreducible polynomials. In [6] Ernest *et al.* reported a reconfigurable implementation of a coprocessor for ECC. The coprocessor is based on two scalable architectures for finite field computations. One of them is a combinatorial Karatsuba multiplier, the other is a finite field coprocessor that implements field multiplication, addition and squaring in hardware. ECC software implementation performance can be improved by using these two finite field coprocessors. If only the first architecture is used, scalar multiplication can be performed in 184 ms. If both architectures are used, the required time reduces to 10.9 ms. In [7], Ernest *et al.* reported a hardware architecture to perform scalar multiplication over the binary field $F_{2^{270}}$ using projective coordinates. Field elements are represented in normal basis, for that reason a Massey-Omura multiplier is used. Coprocessor architecture consist of three basic blocks: a register file with sixteen 270-bit registers, a finite sate machine implementing the doubling and add method to perform scalar multiplication and an arithmetic unit performing field computations. The design was synthesized for a FPGA device equivalent to 180,000 gates. The coprocessor occupies 82% of the available CLBs while performing scalar multiplication in 6.8 ms.

All the reported hardware architectures are related with the arithmetic required for elliptic curve operations. None of them implements an ECC algorithm such as digital signature. Although software implementations have been reported, they are slower than hardware solutions and are unattractive if cryptosystem needs to perform a lot of cryptographic operations, for example, in a secure web server.

3. The elliptic curve cryptosystem

Cryptography can be divided into two groups: symmetric (SKC) and asymmetric key (PKC) cryptography [20]. In the former, the same key is used for encrypting and for decrypting. Symmetric key cryptosystems have been used to provide confidentiality for years. A major disadvantage in symmetric cryptography is the key distribution; the problem complicates when the cryptosystem has to support several users on a dynamic manner. To solve this weakness, PKC was introduced and, together with it, the concept of digital signature. In PKC, two keys are used; a public key is employed to encrypt data while a different, but mathematically related private key is employed to decrypt it. ECC belongs to the algorithms based on PKC, similar to other public key cryptosystems; its security is based on a hard mathematical problem: the elliptic curve logarithm discrete problem ECLDP [1].

An elliptic curve cryptosystem consists of a *7-tuple* $T = (q, FR, a, b, G, n, h)$ where q represents the finite field, FR indicates the representation basis of the finite field, a and b are elements of the finite field F_q and define the elliptic curve equation, G is a point of the elliptic curve and has the property of generating all other points defined by the same elliptic curve, n is the order of the point G and h is the divider of the number of elements of the elliptic curve by n. For details of ECC and its implementations see [1] and [2]. Before implementing an ECC system, these parameters need to be selected. The selections are influenced by security considerations, application platform and, possible design constrains. First we have to decide which field we want to work on. For practical applications, only two finite fields can be used: prime finite field or two-characteristic finite field [2]. The finite field selected does not affect the security of the cryptosystem just the performance of the arithmetic units. It has been shown that binary finite fields lead to better efficient implementation than prime finite fields [2]. For a binary field, an elliptic curve is defined as a set of points satisfying equation (1).

$$y^2 + xy = x^3 + ax^2 + b \tag{1}$$

For specific elliptic curves defined over a finite field the ECPLD is intractable [1]. NIST [15] has recommended some curves that have been proved to be secure to use in practical applications.

After selecting the finite field, the basis representation of the field elements needs to be selected. Basis representation often determines the type of field multiplier to be used. For the binary finite field, polynomial or normal basis can be selected. If polynomial basis is selected, a serial multiplier is used, on the other hand, if normal basis is selected, parallel multiplier (Masey-Omura) is used. Serial multipliers consume fewer resources than parallel multipliers, but require more time to perform a field multiplication. Another important aspect to consider is related to the coordinates for representing points of the elliptic curve. It has been shown in the literature that projective coordinates instead of affine coordinates avoid field inversion which improves considerably the performance of the cryptosystem [2].

Cryptographic schemes based on elliptic curves have been proposed [9]. The three schemes implemented in this paper are explained in detail in the next section.

4. Elliptic Curve Cryptographic schemes

We have selected the following cryptographic algorithms: the Elliptic Curve Diffie-Hellman (ECDH) scheme, the Elliptic Curve Digital Signature Algorithm (ECDSA) and the Elliptic Curve Integrated Encryption Scheme (ECIES). The ECDH is the elliptic curve version of the Diffie Hellman key generation method; the ECDSA is the elliptic curve analogue of DSA; and the ECIES scheme for encryption/decryption has been proposed by the Standards for Efficient Cryptography Group (SECG).

For the following algorithms, the *7-tuple* T is shared by entities A and B, d_A and d_B are private keys of entities A and B respectively and Q_A and Q_B are the public keys of A and B respectively.

4.1. DH key generation

A uses the following procedure to calculate a secret shared value with B:

1. Compute $P = d_A Q_B = (x, y)$
2. $z = x$
3. The secret value is z

z is the shared key between the two entities, A and B.

4.2. ECIES scheme

The ECIES scheme employs two algorithms: a symmetric cipher E and a MAC (Message Authentication Code) algorithm. Assume S is the key's length for the cipher algorithm and M is for the MAC algorithm. A sends an encrypted message m to B executing the following steps:

1. Select a random number k from $[1, n-1]$
2. Compute $(x, y) = kQ_B$ and $R = kG$
3. Derive a $(S + M)$-bit key k_{KDF} from x according to [19].
4. Derive a S-bit key k_S from K_{KDF} and encrypt the message. $C = E(m, k_S)$
5. Derive a M-bit key k_M from K_{KDF} and compute the m's MAC value. $V = MAC(m, k_M)$
6. Send (R, C, V) to B

To recover the original message, B does the following:

1. If R is not a valid elliptic curve point, fail and return.
2. Compute $(x', y') = d_B R$
3. Derive a $(S + M)$-bit key k_{KDF} from x' according to [19].
4. Derive a S-bit key k_S from K_{KDF} and decrypt the message C. $m_1 = E(C, k_s)$
5. Derive a M-bit key k_M from K_{KDF} and compute the m_1's MAC value. $V_1 = MAC(m_1, k_M)$
6. Accept message m_1 as valid if and only if $V = V_1$

Some comments about the required operations for this scheme is given at the end of the next subsection.

4.3. ECDSA scheme

To sign a message, entity A does the following:

1. Select a random number k from $[1, n-1]$
2. Compute $R = kG = (x, y)$ and $r = x \bmod n$. If $r = 0$ go to step 1.
3. Compute $s = k^{-1}(H(m) + d_A r) \bmod n$, H is the hash value of the message.
4. The digital signature on message m is the pair (r, s)

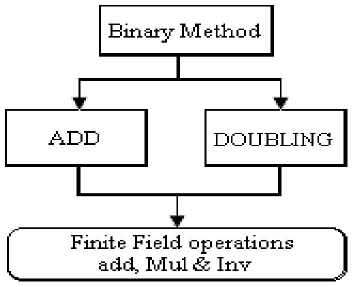

Figure 1. Scalar Multiplication Hierarchy

Entity B can verify the digital signature (r, s) on m performing the following steps:

1. Verify r and s are integers in $[1, n-1]$, if not, the digital signature is wrong. Finish and reject the message.
2. Compute $w = s^{-1} \bmod n$ and $H(m)$, H is the hash value of the message.
3. Compute $u_1 = H(m)w \bmod n$ and $u_2 = rw \bmod n$
4. Calculate $R' = u_1 G + u_2 Q_A = (x', y')$
5. Compute $v' = x' \bmod n$, accept the digital signature if and only if $v' = r$

Each one of the cryptographic algorithms listed above require at least one scalar multiplication, such operation consists of multiplying an integer number k by an elliptic curve point P. This operation can be performed by applying one of the methods reported in [2], [8].

Invariably, scalar multiplication requires several elliptic curve points *Adds* (when two points are equal the add operation is called *Doubling*); at the same time, an *Add* operation requires several field operations. The number and type of these field operations depend on the type of coordinates being used. In the binary method, scalar multiplication is achieved by performing M *Doubling* and N *Add* operations. Here, M is the size of the finite field and N is the number of '$1s$' in the binary representation of k. It has been shown that the use of projective coordinates eliminates the need of performing inversion in each elliptic curve operation and it is a very good alternative to reduce the execution time required for the scalar multiplication. In the cryptosystem we propose, the scalar multiplication is performed according to the hierarchy of operations shown in figure 1. The number and type of field operations required in each elliptic curve add in both affine and projective coordinates are listed in table 1. In table 1, add refers to a sum operations of field elements while *Add* refers to a sum operation of elliptic curve points. Scalar multiplication can be improved even more if NAF representation [8] of the integer operand k is used.

Operation	Affine		Projective	
	Add	Double	Add	Double
add	6	8	4	6
Mul	3	3	10	20
Inv	1	1	0	0

Table 1. Field operations when affine and projective coordinates are used

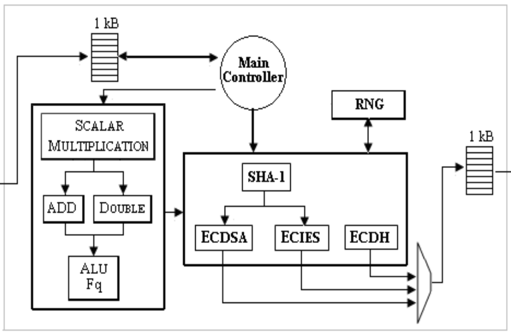

Figure 2. Elliptic Curve Cryptosystem

5. Design and implementation of the elliptic curve cryptosystem

A block diagram of the cryptosystem developed is shown in figure 2. The cryptosystem consists of several sub modules that perform each one of the cryptographic algorithms listed above. The main blocks of the cryptosystem are: a coprocessor that performs the scalar multiplication, an m-bit random number generator module RNG, three modules executing the cryptographic algorithms ECDSA, ECIES and ECDH, and a main controller orchestrating the data flow.

In the cryptosystem, the data flow is as follows: the main controller stores the incoming data going to be encrypted into a 1KB memory, then, according to the required operation, it enables any of the cryptographic modules. Both ECDSA and ECIES modules request a random number to the RNG module, enable SHA-1 module to get the hash value of the incoming data and request to the coprocessor the necesary scalar multiplications. The modular arithmetic operations needed in ECDSA are performed internally by an specialized big-integer ALU. Finally, the encrypted data are stored in a 1KB output memory. If only a secret value for key exchange is required, input ECDH's parameters are entered instead of data and the secret shared value is placed in the output memory.

The ECIES module employs the SHA-1 algorithm to derive the required key in the encryption and decryption procedures described in section 4. In this case, the SHA-1 module generates the key by iterating several times. The encryption algorithm consist of an XOR operation between the original data and the key generated.

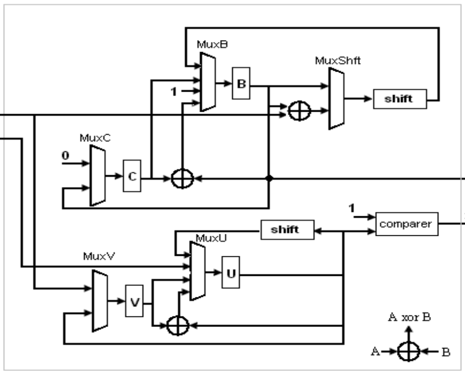

Figure 3. F_{2^m} Inverter

When a digital signature generation or verification is been performing, the SHA-1 module is required only once to generate a 160-bit hash value. The integer multiplication is performed serial executing modular reduction interleaved. The integer inversion, which is more expensive than multiplication, is performed as described in [3].

The random number generator is implemented as a linear feedback shift register (LFSR) [20].

The cryptosystem can operate with elliptic curves over any binary polynomial finite field F_{2^m}. The arithmetic coprocessor is implemented according to the hierarchy of operations shown in figure 1. At the top level of the figure, scalar multiplication is performed according to the binary method, implemented as a finite state machine; *Add* and *Doubling* operations are implemented as finite state machines too. This selection was because this algorithm is too simple to implement and it allows to perform an *Add* and *Doubling* operation in parallel.

The field arithmetic unit consists of a field serial multiplier and an inverter. The inverter is based on the Modified Almost Inverse Algorithm [8], this module dominates the time execution in both *Add* and *Doubling* operations. A diagram of such inverter is shown in figure 3. For the field multiplication, we have implemented a serial shift and add multiplier with interleaved polynomial reduction. The main advantage of using a serial multiplier is that it consumes fewer resources compared with other approaches. A diagram of the architecture of this multiplier is shown in figure 4. A field multiplication can be achieved in m clock cycles by performing field add and shift operations.

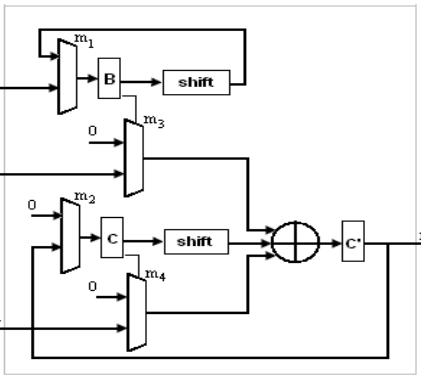

Figure 4. F_{2^m} Multiplier

Module	Slices	Frequency (MHz)
Multiplier	%8	166
Inverter	% 35	115

Table 2. Synthesis results

6. Results

We prototyped the cryptosystem in a Xilinx VirtexII XC2V1000-4FG456 FPGA. Elliptic curve was defined over the finite field $F_{2^{191}}$ and irreducible polynomial $F(x) = x^{191} + x^9 + 1$. The arithmetic coprocessor was described using VHDL. This module was simulated using Active-HDL and synthesized using the Xilinx ISE software. Synthesis results are shown in table 2. The full system was verified co-simulating the Handel C and VHDL code.

In table 3 we show the timing results obtained for elliptic curve and field arithmetic. The scalar multiplication is performed in 32 ms if affine coordinates are used applying the binary method. If projective coordinates and a NAF representation of the integer operand are used, scalar multiplication can be performed in 4.7 ms.

A performance comparison of hardware implementations against each other is not straight forward. It is because of different key size and FPGA technology used for their implementation. In table 4, the scalar multiplication timing result we have obtained is compared with some hardware implementations mentioned earlier in this paper.

Operation	Time (ms)	
	Affine	Projective
Scalar mul	32	4.7
Mul	0.00148	
Inv	0.135	

Table 3. Timing results

Reference	F_q	Platform	Time (ms)
This work	$F_{2^{191}}$	Xilinx XC2V1000	4.7
[16]	$F_{2^{163}}$	Altera EPIF10K250	80
[6]	$F_{2^{113}}$	Amtel AT94K40	10.9
[7]	$F_{2^{270}}$	Xilinx XC4085XLA	6.8

Table 4. Timing comparison for scalar multiplication

7. Conclusions

We described the design of an elliptic curve cryptosystem over F_{2^m}. This cryptosystem is able to execute any of three cryptographic schemes (key exchange, encryption/decryption and digital signature). The cryptosystem has been implemented using hardware description languages and tested on a Virtex II FPGA device. This cryptosystem performs a fast elliptic scalar multiplication, which is the most time consuming part in each of the cryptographic algorithms . For a 191-bit scalar multiplication, operating to 113 MHz, it takes 4.7 ms if projective coordinates are used. The designed arithmetic coprocessor supports different binary polynomial fields and can be further improved to perform faster scalar multiplication.

8. Future Work

We are working currently on reducing the time needed to perform scalar multiplication, mainly by using a field digit serial multiplier and applying the Mongomery method [8]. Theoretically, if a 16-digit serial multiplier is used; scalar multiplication can be performed in 0.2 ms.

Also, we have identified that the cryptosystem throughput can be improve if data is compressed before. So We are developing a lossless data compressor module that can operate jointly with the developed cryptosystem.

References

[1] American Bankers Association. ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

[2] R. Dahab and J. López. An Overview of Elliptic Curve Criptography. Technical Report, IC-00-10, http://citeseer.nj.nec.com/333066.html.

[3] A. Daly *et al.* Fast Modular Division for Application in ECC on Reconfigurable Logic. In *Proc. of 13th International Conference on Field Programmable Logic and Application, FPL'2003*, volume 2778 of *Lecture Notes in Com-*

puter Science, pages 786–795, Lisbon, Portugal, September 2003. Springer.

[4] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(2/3):644–654, November 1976.

[5] H. Eberle, N. Gura, and S. Chang. A Cryptographic Processor for Arbitrary Elliptic Curves over GF(2^m). In *Proc. of IEEE 14th International Conference on Application-specific Systems, Architectures and Processors,ASAP'2003, The Hague, The Netherlands*, pages 444–454, June 2003.

[6] M. Ernest *et al.* A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2^n). In *Proc. of the 4th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 381–399, Redwood Shores, CA, August 2002. Springer.

[7] M. Ernest *et al.* Rapid Prototyping for Hardware Accelerated Elliptic Curve Public Key Cryptosystems. In *Proc. of 12th IEEE Workshop on Rapid System Prototyping, RSP'2001*, pages 24–31, Monterey, CA, June 2001.

[8] D. Hankerson, L. López, and A. Menezes. Software Implementation of Elliptic Curve Cryptography over Binary Fields. In *Proc. of the Second International Workshop on Cryptographic Hardware and Embedded Systems, CHES'2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 1–24, Worcester, MA, August 2000. Springer.

[9] IEEE P1363 Committee. Standards Specification for Public key Cryptography, http://grouper.ieee.org/groups/1363/.

[10] T. Kerins *et al.* Fully Parameterizable Elliptic Curve Cryptography Processor over GF(2^m). In *Proc. of 12th International Conference on Field Programmable Logic and Application, FPL'2002*, volume 2438 of *Lecture Notes in Computer Science*, pages 750–759, Montpellier, France, September 2002. Springer.

[11] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, November 1987.

[12] N. Koblitz, S. Vastone, and A. Menezes. The State of Elliptic Curve Cryptography. *Designs, Codes and Cryptography*, 19(2/3):173–193, March 2000.

[13] E. Konstantinou, Y. Stamatiou, and C. Zaroliagis. Software Library for Elliptic Curve Cryptography. In *Proc. of 10th Annual European Symposium on Algorithms, ESA'2002*, volume 2461 of *Lecture Notes in Computer Science*, pages 625–637, Rome, Italy, September 2002. Springer.

[14] V. Miller. Use of Elliptic Curves in Cryptography. In *Proc. of Advances in Cryptology, CRYPTO'85*, pages 417–426, Santa Barbara, CA, August 1985.

[15] NIST. Recommended Elliptic Curves for Federal Government Use, http://csrc.nist.gov/csrc/fedstandards.html.

[16] S. Okada *et al.* Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2^m) on a FPGA. In *Proc. of the Second International Workshop on Cryptographic Hardware and Embedded Systems, CHES'2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 25–40, Worcester, MA, August 2000. Springer.

[17] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m). In *Proc. of the Second International Workshop on Cryptographic Hardware and Embedded Systems, CHES'2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 41–56, Worcester, MA, August 2000. Springer.

[18] A. Satoh and K. Takano. A Scalable Dual-Field Elliptic Curve Cryptographic Processor. *Transactions on Computers*, 52(4):449–460, April 2003.

[19] SEC1. Elliptic Curve Cryptography: Standards for Efficient Cryptography Group, http://www.secg.org.

[20] W. Stallings. *Cryptography and Network Security*. Prentice Hall, NJ, 2003.

[21] A. Weimerskirch, C. Paar, and S. Chang. Elliptic Curve Cryptography on a Palm OS Device. In *Proc. of the 6th Australasian Conference, ACISP'2001*, volume 2119 of *Lecture Notes in Computer Science*, pages 502–513, Sydney, Australia, July 2001. Springer.

IEEE COMPUTER SOCIETY