IMPLEMENTATION AND PERFORMANCE ANALYSIS OF ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM

Mohammad Noor Nabi, Sabbir Mahmud, and M Lutfar Rahman*

School of Engineering and Computer Science, Independent University, Bangladesh

* Department of Computer Science and Engineering, University of Dhaka, Bangladesh

E-mail: mnnabi@iub.edu.bd, smahmud@iub.edu.bd, lrahman@udhaka.net

Abstract: *In this paper we present a novel method for obtaining fast software implementation of the Elliptic Curve Digital Signature Algorithm in the finite field GF(p) with an arbitrary prime modulus p of arbitrary length. The most important feature of the method is that it avoids bit-level operations which are slow on microprocessors and performs word-level operations which are significantly faster. The algorithms used in the implementation perform word-level operations, trading them off for bit-level operations and thus resulting in much higher speeds. We provide the timing results of our implementations on a 2.8 GHz Pentium 4 processor, supporting our claim that ECDSA is appropriate for constrained environments.*

Keywords: *Digital Signature, Elliptic Curve, Finite Field, ECDSA, Hashing.*

1. Introduction

A digital signature is a checksum which depends on the time period during which it was produced. It depends on all the bits of a transmitted message, and also on a secret key, but which can be checked without knowledge of the secret key.

Digital signatures furnish the parties with two forms of protection, where A is the originator and B the receiver of message. 1) Both party A and party B should be protected against forged messages, planted in communication system by a third party C who pretends to be party A. 2) Party A should be protected against messages forged by party B, who may claim to received the messages from party A.

At this time, there are three popular public-key algorithms which can provide digital signatures: (1) Elliptic Curve Digital Signature Algorithm (ECDSA) [1]; (2) the RSA scheme [2], (3) the ElGamal signature scheme [3]. Among these ECDSA provides a faster alternative for public-key cryptography, much smaller key lengths are required to provide a desired level of security [1].

The objectives of this paper are to implement Elliptic Curve Digital Signature Algorithm (ECDSA) in C/C++ and analyze its workload characteristics. This paper also finds the suitability of ECDSA in constrained environment where the processing resources, memory and power are all very limited.

In Section 2 some of the related works are mentioned. Section 3 describes briefly about elliptic curve cryptography and generation of parameters of ECDSA. Section 4 describes the ECDSA. Section 5 describes the algorithms used in the implementation of ECDSA. In section 6 experimental results are presented. Section 7 concludes the paper.

2. Previous Work

Elliptic curves as mathematical objects have been known and studied since long before digital computers were built, but their application in cryptography has been more recent. In 1985, Victor Miller [4] and Neal Kolbitz [5] suggested independently that elliptic curves could he used to perform public-key security functions (e.g. key exchanges, digital signatures). Detailed and comprehensive reference is available on techniques for efficient finite field and elliptic curve arithmetic is IEEE 1363-2000 [6]. Elliptic-curve ElGamal (EC-EIGamal) is the elliptic-curve analog of the integer ElGamal algorithm [7]. ECC provide the highest strength per bit among cryptosystems known today [8]. Jin-Hee Han and al implemented ECDSA based on Java card [9]. T.Yanik and al implemented ECDSA by incomplete reduction in modular arithmetic [10].

3. Elliptic Curve Cryptography

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA). It was accepted in 1999 as an ANSI standard, and was accepted in 2000 as IEEE PI363 [10] and NIST’s FIPS 186-2 [12] standards. It was also accepted in 1998 as an ISO standard, and is under consideration for inclusion in some other ISO standards. Unlike the ordinary discrete logarithm problem and the integer factorization problem, no subexponential-time algorithm is known for the elliptic curve discrete logarithm problem. For this reason, the strength-per-key-bit is substantially greater in an algorithm that uses elliptic curves.

3.1 Comparison of Elliptic Curve Cryptography

Certain unique properties of elliptic curves made them resilient against the types of attacks that were successful against integer-based algorithms. Table 1 shows, the number of key bits necessary to have equivalent levels of security for integer based algorithms (e.g. Digital Signature Algorithm, RSA) versus elliptic curve algorithms (e.g. ECDSA). For the most commonly used 1024-bit keys for an integer based algorithm; the elliptic curve counterpart only requires 160-bit keys for the equivalent security [11]. This is 7 times reduction in the space required to store these keys, or a similar reduction in bandwidth required to transmit these keys over a wireless network. This reduction in the size of data objects allows much faster completion of the algorithms. Because of these favorable properties, ECC has been incorporated into many security standards.

Table 1. Key lengths (in bits) for equivalent security

Integer algorithm (e.g. DSA, RSA)	Elliptic curve algorithm (e.g. ECDSA)
512	106
768	132
1024	160
2048	210
21000	600

3.2. Elliptic Curve Parameters

Implementation of elliptic curve cryptography involves the selection of a suitable elliptic curve (determined by the coefficients in the elliptic curve equation), the representation of field elements (e.g. a binary field or a prime field), algorithms for field arithmetic and elliptic-curve arithmetic. The standards provide suggestions for the selection of elliptic curves and representation of field elements. FIPS 186-2 [12] recommends a total of ten curves for binary fields: two different curves for each of 163-bit, 233-bit, 283-bit, 409-bit and 571-bit fields. We limit the scope of this analysis to five curves on binary fields, and choose polynomial basis representation for the field elements, which allows faster implementation on programmable processors.

4. Elliptic Curve Digital Signature Algorithm

Elliptic-curve ElGamal (EC-EIGamal) is the elliptic-curve analog of the integer ElGamal algorithm [13]. It is used to securely transmit the coordinates of the point P(x, y) from party A to party B (assume that the original plaintext *m* is embedded in P(x, y). We assume that party A and party B have previously agreed on a binary field GF(2^k), a common elliptic curve *E* with suitable coefficients, and a base point, which lies on *E* and has order ***n*.**

Elliptic-curve Digital Signature Algorithm (EC-DSA) has three different segments: key generation, signature generation and signature verification. These steps are summarized in Figure 1, where party A signs the message *m* and party B verifies the signature.

```
Key generation (by party A)
1. Choose random $a \in [2, n-1]$
2. Compute intermediate point $A_T$
        $A_T = P \times a$
        Party A's private key = $a$
        Party A's public key = $(E,P,A_T)$
Signature Generation (by party A)
1. Choose random $k \in [2, n-1]$
2. Compute $P \times a = (x_1, y_1)$
   and
   $R = x_1 \bmod n$ ( if r = 0, go to step 1)
3. Compute $K^{-1} \bmod n$
4. Compute
   $s = k^{-1}(SHA(m)+ar) \bmod n$
   (if s = 0, go to step 1)
   Signature for m = (r.s)
5. send (r,s)
--------------------------------------------
Signature verification (by party B)
--------------------------------------------
1.  Compute $c = s^{-1} \bmod n$
    and
    SHA(m)
2.  Compute $u_1$= SHA(m)c mod n

    And $u_2$= rc mod n
3.  Compute
    $P \times u_1 + A_T \times u_2 = (x_0, y_0)$
    and
    $V = x_0 \bmod n$
4. Accept signature if v = r
```

Figure 1. Elliptic Curve Digital Signature Algorithm

5. Software Implementation Of ECDSA

ECDSA is coded and optimized in C/C++ on IBM workstation using 2.8 GHz Intel Pentium 4 processor. The coding of these algorithms needs fairly simple instructions, but efficient algorithms mentioned in section 4 were used.

The basic arithmetic operations (i.e. addition, subtraction and multiplication) in the finite field GF(p) have several applications in cryptography, including Elliptic Curve Digital Signature Algorithm (ECDSA) [14].

The arithmetic of GF(p) is also called modular arithmetic where the modulus is p. The elements of the field are the set of integers {0, 1, . . ., (p-1)}, and the arithmetic function (addition, subtraction and multiplication) takes two input operands from this set and produces the output which is also in this set. We are assuming that the modulus p is a k-bit integer, where $k \in [160, 2048]$. A number in this range is represented as an array of words, where each word is of length w. Most software implementations require that w = 32; however, w can be selected as 8 or 16 on 8-bit or 16 bit microprocessors. Algorithms used in the implementation of ECDSA are shown in figure 2 and figure 3.

Algorithm 1 is used for polynomial reduction in binary fields. Algorithm 2 is used for polynomial reduction in binary fields. This is facilitated by using 512-byte table that is pre-computed to hold 16-bit squares of each 8-bit polynomial [14]. For polynomial inversion, we present Modified Almost Inverse Algorithm (MAIA) [14] which is summarized in Algorithm 2. MAIA (and similar variants of the Almost Inverse Algorithm) is used in optimized implementations. Algorithm 4 was used for adding two points and Algorithm 5 was used doubling of a point on elliptic curves.

Nomenclature for algorithm descriptions: Polynomials are represented using lower-case letters: *a*(x), *b*(x), *c*(x) etc. When addressing the individual 64bit words of a polynomial, square brackets are used: *a*[0], *b*[l], c[2] *etc. a*[0] represents the lowest order (least significant) word of a(x). When addressing the individual bits of a polynomial, a subscript is used: a_0, b_{32}, c_{162} etc. The bit a_0 represents the least-significant bit of *a*(x), a_{162} represents the most-significant bit. The operator $\oplus$ represents an XOR operation. When used, *p*(x) denotes the irreducible polynomial generating the field. For $GF(z^{163})$, $p(x) = x^{163} + x^7 + x^4 + x^3 + 1$.

```
Algorithm 1. Polynomial reduction [8]
INPUT: Binary polynomial $c(x)$ of degree at most 324.
OUTPUT: $c(x) \bmod p(x)$, where $p(x) = x^{163} + x^7 + x^6 + x^3 + 1$
1. For $i$ from 5 down to 3 do
        1.1 $t = c(i)$ .
        1.2 $c(i-3) \equiv c(i-3) \oplus (t<<29) \oplus (t<<32) \oplus (t>>35) \oplus (t>>36)$
        1.3 $c(i-2) \equiv c(i-2) \oplus (t<<28) \oplus (t<<29) \oplus (t>>32) \oplus (t>>35)$
2. $t = c(i)$&0xFFFFFFFF800000000.
3. $c(0) \equiv c(0) \oplus (t<<28) \oplus (t<<29) \oplus (t>>32) \oplus (t>>35)$
4. $c[2] = c[2]$&0x00000007FFFFFFFF.
5. Return ($c[2], c[1], c[0]$).
```

```
Algorithm 2. Table lookup method for polynomial squaring
INPUT: Binary polynomial $a(x)$
OUTPUT: $C(X) = a^2(x)$
```

```
1. Precomputation: For each byte $v = (v_7, v_6 ... v_1, v_0)$.
compute the 16-bit quantity $T(v) = (0, v_7, 0, v_6 ... , 0, v_1, 0, v_0$
2. For $i$ from 0 to 5 do
    2.1. Let $a[il = ( u_7, u_6, u_5, u_4, u_3, u_2, u_1, u_0)$ where each $u_i$ is a byte.
    2.2. $c[2i]=(T(u_1), T(u_0)), c[2i+1]=( T(u_3), T(u_2))$
3. Return c(x).
```

```
Algorithm 3. Modified Almost Inverse Algorithm (MAIA) [8,23] for polynomial inversion
INPUT: Binary polynomial $a(x)$, $a(x) \neq 0$
OUTPUT:
$b(x) \in GF(2^t)$ and $t \in [0,2k-1]$ Such that $b(x)\,a(x) \equiv x^t \bmod p(x)$
1. $b(x)=1, c(x)=0, u(x)=a(x), v(x)= p(x), t=0$.
2. While $x$ divides $u(x)$ do
        2.1 $u(x)= u(x)/x, c(x)= c(x)x, t = t+1$
3. $u(x) = 1$, return $(b(x)t$.
4. If degree $(u(x))<$degree$(v(x))$ then $u(x) \leftrightarrow v(x), b(x) \leftrightarrow c(x)$.
5. $u(x)= u(x)+ x^j v(x), b(x)= b(x)+ c(x)$
6. Go to Step 2.
```

Figure 2. Algorithms for (a) polynomial reduction, (b) polynomial squaring (c) Polynomial inversion

```
Algorithm 4. Adding two distinct points on an elliptic curve
INPUT: Elliptic Curve points $P =(x_1, y_1)$ and $Q =(x_2, y_2)$, $P \neq Q$
OUTPUT: $R = P + Q = (x_3, y_3)$
    1. Compute $\theta = \frac{y_2 + y_1}{x_2 + x_1}$
    2. Compute $x_3 = \theta^2 + \theta + x_1 + x_2 + a$
    3. Compute $y_3 = \theta(x_1 + x_3) + x_3 + y_1$
    4. Return $(x_3, y_3)$.
```

```
Algorithm 5. Doubling a point on an elliptic curve
INPUT: Elliptic Curve point $P =(x_1, y_1)$
OUTPUT: $R = P + P = (x_3, y_3)$
    1. Compute $\theta = x + \frac{y}{x}$
    2. Compute $x_3 = \theta^2 + \theta + a$
    3. Compute $y_3 = \theta(x^2 + (\theta + 1)x_3$
    4. Return $(x_3, y_3)$.
```

Figure 3. Algorithms for (a) adding points (b) doubling points on an elliptic curve

Separate program was written to count points on the curve and specify an initial point on the curve using algorithms shown in figure 3, which was then used in a separate program as common information to generate public and private key. The source codes were carefully minimized by creating separate functions to handle polynomials using different algorithms shown in figure 2 and figure 3. Separate main programs were written for signature generation and verification of ECDSA described in section 4 where algorithms shown in figure 2 were used.

6. Experimental Results

Experiments were performed on many different sets of data. For characteristic representative of workload characterization, we used the three different sets of inputs in the range of 2000000 to 100000000 characters without space. The experiments were conducted on an IBM workstation with Intel processor Pentium 4 of 2.8 GHz clock speed, memory size 512 MB, cache memory size 512 KB and storage of 40 GB. Table 2 shows the hashing time obtained from the experiments. The number of characters is represented by ***n*,** and the times taken (in milliseconds) by hashing is stated by $\boldsymbol{T_H}$**.** Figure 4 shows that the 'hashing' time increases linearly with the number of characters or block size. Hashing is very fast as we found that for 10 million characters without spacing it takes around 406 milliseconds.

Table 3 and Table 4 shows the time obtained from our experiments for five different key sizes. The key sizes were taken for five different curves. The key sizes were selected according to equivalency shown in Table 1. In these tables T_{GS1}, T_{GS2}, T_{GS3}, T_{GS4}, and T_{GS5} represent the times taken (in milliseconds) for signature operations using key sizes of 106,132, 160, 224 and 512 bits respectively. T_{SWH1}, T_{SWH2}, T_{SWH3}, T_{SWH4}, and T_{SWH5} represent the times taken (in milliseconds) for signature operations without hashing and T_{VS1}, T_{VS2}, T_{VS3},T_{VS4} and T_{VS5} represent the times taken (in milliseconds) for signature verification operations for the above mentioned five key sizes respectively. The times T_{GS5} and T_{VS5} taken for digital signature generation and verification using key size of 512-bits are considered to calculate the speedup.

Table 2. Experimental results for hashing using SHA1

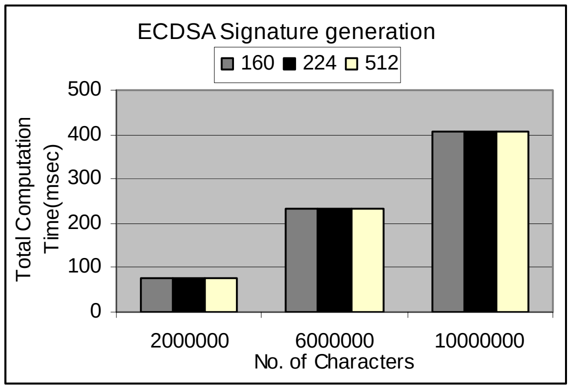
N	T_H (msec)
2000000	78
6000000	234
10000000	406

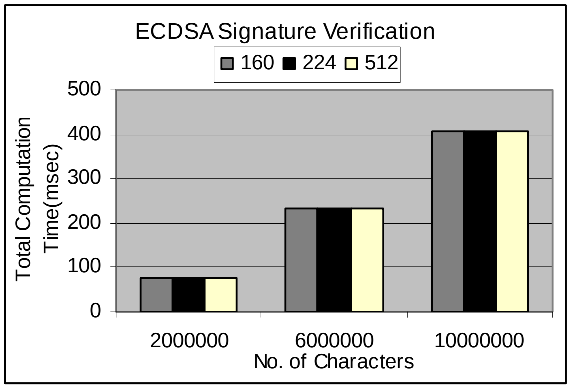
Table 3. Experimental results of ECDSA for key size 106, 132 and 160 bits

n	Key Size: 106			Key Size: 132			Key Size: 160		
	T_{GS1} (msec)	T_{SWH1} (msec)	T_{VS1} (msec)	T_{GS2} (msec)	T_{SWH2} (msec)	T_{VS2} (msec)	T_{GS3} (msec)	T_{SWH3} (msec)	T_{VS3} (msec)
2000000	78	0	78	78	0	78	78	0	78
6000000	234	0	234	234	0	234	234	0	234
10000000	406	0	406	406	0	406	406	0	406
Speedup	1	-	1	1	-	1	1	-	1

Table 4. Experimental results of ECDSA for key size 224 and 512

N	Key Size: 224			Key Size: 512		
	T_{GS1} (msec)	T_{SWH1} (msec)	T_{VS1} (msec)	T_{GS2} (msec)	T_{SWH2} (msec)	T_{VS2} (msec)
2000000	78	0	78	78	0	78
6000000	234	0	234	234	0	234
10000000	406	0	406	406	0	406
Speedup	1	-	1	1	-	1





(a) **(b)**

Figure 4. Result analysis of the results for ECDSA, (a) Signature generation time; (b) Signature Verification time; using key sizes 160, 224 and 512 bits respectively

Figure 5 states the relationship between the results from our experiments. Figure 4(a) show signature generation and Figure 4(b) show signature verification time respectively using key size 160 bits, 224 bits and 512-bits. It is found that for the similar number of characters (2000000 and 10000000) ECDSA takes similar amount of time for input of same sizes. It is found that for ECDSA signature generation and verification times are equivalent to their hashing times respectively. Table 3 and Table 4 show that the signature generation without hashing is 0 in millisecond scale, which means the operation needs less than 1 millisecond for five key sizes considered in the experiment. In millisecond scale we did not find any speed up although key sizes were varied from 106 to 512 bits which means that Speedup factor does not increase with key size. ECDSA is very fast, signature generation and verification time is very insignificant in comparison with hashing time.

7. Conclusions

In this paper we presented practical implementation of ECDSA signature generation and verification algorithms and found its workload characteristics. ECDSA can provide very high speed signature generation and verification. As much smaller key length is required with ECDSA to provide desired level of security, key exchanges become faster and smaller key storage is needed. ECDSA is therefore much better than DSA and RSA signatures for constrained environment like mobile information appliances, where computing resources and power ability are limited. ECDSA can be used equally in non-constrained environments. We hope that this paper contributes to an increased understanding of the properties of ECDSA, and facilitates its use in practice.

For future work, we plan to expand our research to include other digital signature algorithms, elliptic-curve algorithms, different block ciphers, symmetric-key, and hash functions. We will also expand our ECC results to include results for prime fields, using bases other than polynomial bases, and different coordinate systems such as projective coordinates.

References

[1] Araki. Kiyomichi, Takakazu Satoh, and Shinji Miura, "Overview of Elliptic Curve Cryptography," *Public Key Cryptography*. pp. 2948. *Springer-Verlag*. 1998.

[2] Rivest, R.L., Shamir, A., and Adelman, L. "A method for obtaining digital signatures and public-key cryptosystem", *Commun. ACM,* 1978, 21, *(2).* pp. 120-126.

[3] Elgamal, T, "A public-key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Trans. Info. Theo,* 1985, IT-31, pp. 469-472

[4] Miller. Victor S., "Use of Elliptic Curves in Cryptography," *Lecture Notes in Computer Sci. no. 218, pp. 417.426,* Springer-Verlag. 1986.

[5] Koblitz, Neal. "Elliptic Curve Cryptosystem" *Mathematics of Computation, vol 48, no, 177, pp203-209,* 1987

[6] T. Hasegawa, J. Nakajima and M. Matsui, "A practical implementation of elliptic curve cryptosystems over GF(p) on a 16-bit microcomputer", *Public Key Cryptography – Proceedings of PKC '98*, Lecture Notes in Computer Science, **1**431 (1998), 182-194.

[7] "Proposed Federal Information Processing Standard for Digital Signature Standard (DSS)", in *Federal Register,* 1991, 56, (169),

[8] Menezed, A., Van Oorschot, P., Vanstone, S., *Handbook of Appl Cryptography,* CRC Press, 1997.

[9] Jin-Hee Han and al, "Implementation of ECC/ECDSA Cryptography Algorithms Based on Java Card", *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02),* 2002 IEEE

[10] T.Yanik, E. Savas and C. K. Koc "Incomplete reduction in modular arithmetic" *IEE Proceedings online no. 20020235,* 2002.

[11] Annex to the IEEE P1363, Standard Specifications for Public Key Cryptography.

[12] "Digital Signature Standard (DSS) - FIPS Pub. 186-2" February 2000.

[13] EIGamal, T., "A Public key Cryptosystem and a Signature Scheme based on Discrete Logarithm," *IEEE Trans. on Info theory, 31:469-472.* 1985.

[14] Hankerson, D., J. Hernandez. and A. Menezes. "Software Implementation of Elliptic Curve Cryptography Over Binary Fields,'' *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES* 2000), 2000.

Mohammad Noor Nabi is currently a Lecturer in the School of Engineering and Computer Science at the Independent University, Bangladesh. Mr. Nabi received his B. Sc. (Honors) in Applied Physics, Electronics and Communication Engineering, M.S. in Computer Science and Engineering degrees from University of Dhaka. Mr Nabi's areas of interest include Parallel and distributed computing, Cryptography and VLSI design.

Sabbir Mahmud is currently a Lecturer in the School of Engineering and Computer Science at the Independent University, Bangladesh. Mr. Mahmud received his B. Sc. (Honors) in Computer Science from IUBAT and M.S. in Computer Science from Independent University, Bangladesh. Mr Mahmuid's areas of interest include Parallel and distributed computing, Cryptography and Data Mining.

M. Lutfar Rahman is currently working as a Professor in the Department of Computer Science and Engineering, University of Dhaka and he is the founder chairman of the Department. He worked as the director of Institution of Information Technology (IIT) of Dhaka University. Professor Rahman obtained B.Sc. (Hons) and M.Sc. in Physics and M.Sc. and Ph.D. in Electronic and Electrical Engineering. He has over 200 research papers and scientific and technical articles to his credit. He authored sixteen books on Electronics and Computer Science and was awarded "Halima Sharfuddin Science Writer Prize" by Bangla Academy.