

Computing Cryptographic Algorithms in Portable and Embedded Devices

Archana Ramachandran, Zhibin Zhou, Dijiang Huang
 { Archana.Ramachandran, Zhibin.Zhou, Dijiang.Huang }@asu.edu

Abstract— Elliptic curve cryptography (ECC) and Pairings get more research attentions for computational constraint devices such as Pocket PCs and wireless sensors. As a result, computational evaluations of various cryptographic algorithms are highly desired to guide researchers to design effective communication protocols. In order to achieve this goal, we conduct comparative performance evaluations for Pocket PC and wireless sensors to study the computational ability to process cryptographic functions, such as point multiplication, Pairings, AES, and hash functions. Our study shows that current Pocket PC level devices are capable to process computational intensive cryptographic functions, such as Pairings. However, purely software cryptographic solutions require long time to process cryptographic algorithms and special optimization methods must be used to improve the computation performance.

Index Terms— Advanced Encryption Standard (AES), Elliptic Curve Cryptography (ECC), Pairings.

I. INTRODUCTION

Portable and embedded devices such Pocket PCs and sensors are being used extensively in our daily life with many appealing features such as portability, mobility, sensing, identifying, tracing, etc. These features are very useful in classifying, counting, and organizing essential objects, including humans' mobility study, personal identification, online transaction, merchandize and so on. Because portable and embedded devices usually carry privacy-critical information, to enhance security is of great importance. Elliptic curve cryptography (ECC) is generally considered as an efficient and effective cryptographic solution for light-weight devices [2,3]. New cryptographic algorithm such as Pairing gets more attentions recently due to its compelling ability in many net-centric applications [6,8]. In many scenarios, cryptography capability of portable and embedded devices is required for confidentiality and data/identity integrity. Thus, a comparative study of computational ability to process cryptographic functions in portable and embedded devices is highly desired.

The goal of this research is to conduct a comparative study for evaluating the performance of cryptographic algorithms such as ECC, Pairings, Hash functions, and AES in computationally restricted devices. Computationally restricted devices are devices in which resources like memory, processor speed etc are limited. Benchmarks are created to identify an efficient security algorithm with device memory and processor speed as constraints to be implemented in these resource restricted devices. ECC, Pairings, hash functions, and AES are chosen for these devices because the key size required for these algorithms is less and/or the computing time is shorter compared to other cryptographic algorithms to achieve the

same degree of security. Among the various computationally restricted devices, the primary focuses of this research are wireless sensors and Pocket PC.

The rest of this paper is arranged as follows: the hardware configuration and testing environments are addressed in section II; we decouple the basic operations of ECC and Pairings in section III; the performance evaluations are given in section IV; in section V, we describe the related work; finally, we summarize our work in section VI.

II. HARDWARE CONFIGURATIONS AND TESTING ENVIRONMENTS

A. Pocket PC

The powerful HP iPAQ hx4700 Pocket PC is a Microsoft Windows Powered Pocket PC designed with a visually stunning VGA color display, a broad range of feature-rich applications, integrated wireless capabilities, enhanced security and dual-slot expansion. The hardware configuration is given in Table 1.

Table 1 Configurations of HP iPAQ HX4700.

Parameters	Specifications
Processor	32 bit-624 MHz Intel Bulverde technology-based RISC processor
OS	Microsoft Windows Mobile 2003 Second Edition software for Pocket PC
Memory	192 MB total memory (128 MB ROM and 64 MB SDRAM) Up to 135 MB user available memory that includes 80 MB iPAQ File Store

Table 2 Hardware Configurations of Micaz 2400.

Model	Micaz
Micro-controller	ATMega128L (8 bit)
No. of processor units	8
Clock Speed	7.37 MHZ
RAM	4 KB
Program Memory	128 KB
Flash	512 KB
Radio Frequency	2400 MHZ

B. Sensors

Wireless sensors are sensors connected in a wireless network consisting of spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at

different locations. A Mote is hardware platform consists of Processor/Radio boards and is typically combined with a sensor of some type to create a remote sensor. These battery-powered devices run micro-power, networking stack, open-source TinyOS operating system which provides low-level events and task management, etc. For the implementation purpose, we select Micaz 2400 as our testing platform. The hardware configurations are given in Table 2.

C. Testing Environments and Settings

1) Testing Pocket PC

We use HP iPAQ hx4700 Pocket PC which runs Microsoft Windows Mobile 2003 operating system. The specification for the Pocket PC is given in section Table 1. The processor in the Pocket PC is a 32 bit processor. The application for the Pocket PC was developed in Microsoft Visual Studio 2005 as a smart device project using a machine running Windows XP. We use a software library from Shamus Software Ltd called Miracl [4] for implementing pairing, hash and encryption algorithms in Pocket PC.

2) Testing Environment Setup for Wireless Sensors

Micaz 2400 wireless sensor running TinyOS was used as a testing environment to calculate the time taken for primitive field operations in pairing as explained in section IV. The sensor is connected to a MIB 510 base station which is in turn connected to the personal computer via an USB port. The code was developed in TinyOS operating system using NESC language. We use the software package TinyECC [5] developed by North Carolina State University research group for implementing primitive field operation in pairing.

III. ECC AND PAIRING ALGORITHMS

In this section, we decouple the ECC and Pairings algorithms into basic field modular operations.

A. Finite Field Arithmetic

ECC is based on the algebraic structure of elliptic curve over finite fields. Thus, the finite field operations are the basis for ECC algorithms. Followings are some operations on a finite field F_p .

- a) Addition: $a+b=c \pmod P$
- b) Multiplication: $a \cdot b=c \pmod P$,
- c) Squaring: $a^2=b \pmod P$,
- d) Inversion: $a=1/b \pmod P$.

Among these operations, we use *modular multiplication* as the basic operations for our performance evaluations.

B. Elliptical Curve Cryptography

In our evaluation, we use the elliptic curve in the form of $y^2 = x^3 + ax + b$ over a *finite field*. The set of points on such a curve (i.e., all solutions of this equation together with a point at infinity) form an *abelian* group (with the point at infinity as identity element). For example, if the coordinates x and y are chosen from a large finite field, the algebraic solutions form a finite *abelian* group. The security of ECC is due to a class of

problems called discrete logarithm problems (DLP) where given 'a' and 'c' in the equation $a^b = c$ it is difficult to compute b . The DLP is believed to be a hard problem. The benefit of ECC is the key size that can be much shorter comparing to other cryptographic algorithms such as RSA for the same comparable level of security. Before talking about how encryption and decryption are done using ECC, we need to understand the primitive operations of ECC such as Point addition and Point doubling.

1) Point Addition and Point Doubling

Elliptic curve groups used in cryptography can be defined over two kinds of fields: $GF(P)$ (GF stands for Galois field), where P is a prime, and $GF(2^m)$ where each element is a binary polynomial of degree m (that can be represented as an m -bit string since each coefficient is either 0 or 1). Suppose the curve equation is $y^2 = x^3 + ax + b$ to add two points, draw a line through them and reflect the third point, where this line intersects the curve, in the x -axis. Algebraically, the result of adding points $A(x_A, y_A)$ and $B(x_B, y_B)$ is $C(x_C, y_C)$ such that $x_C = s^2 - x_A - x_B$, $y_C = -y_A + s(x_A - x_C)$ where $s = (y_A - y_B)/(x_A - x_B)$ is the slope of the line through A and B . When A equals B , the line through A and B degenerates to the tangent at A . This operation is defined as *point doubling* (adding a point to itself). The result of adding A and $-A$ is defined to be a special point called the *point at infinity*. The counts of field operations for *point addition* and *point doubling* is listed in Table 3 which is derived from [3].

Table 3 Counts of basic operations in point addition and point doubling.

Point Addition	Mixed Addition	Point Doubling
$A+A=A$	$J+A=A$	$2A=A$
$P+P=P$	$J+C=C$	$2P=P$
$J+J=J$	$C+A=C$	$2J=J$
$C+C=C$		$2C=C$

A =affine, P =standard projective, J =Jacobian, C =Chudnovsky, I =inversion, M =multiplication, S =squaring

2) Point Iteration

Point iteration is adding a point to itself multiple times. If a point is added to itself $k-1$ times, where k is a positive integer then point iteration is represented as:

$$[k]P = P + P + \dots + P \quad (k-1 \text{ times}).$$

ECC scheme relies on the point iteration to create public key and encrypt message.

C. Pairing Algorithms

Pairing-Based Cryptography has been exploded for several years. The central idea is the construction of a bilinear mapping (see [6] for detailed information of bilinear mapping) between two useful cryptographic groups (ECC points group and OEF), which transfer the properties of one group to another. There are two kinds of known implementations of these pairings – the Weil and Tate pairings. Although first used as cryptanalytic tools, they can also be used for constructive purpose [6].

1) Algorithm (Miller's Algorithm)

Miller algorithm [9] is one of the algorithms to do the above Pairings operation. Let E be an elliptic curve over F_q and let P and Q be given points of prime order l for which we want to compute (P, Q) . The Miller algorithm works as follows:

Miller Algorithm:

Step 1

Choose a random point S element of $E(F_q^k)$ and set $Q_0 = Q + S$ element of $E(F_q^k)$.

Step 2

Set $l = \sum_{i=0}^n b_i 2^i$, $T_1 = P$, $f_1 = 1$.

Step 3

While $n \geq 0$ do

Set $T_1 = [2]T_1$ and $f_1 = f_1^2 (l_1(Q_0)l_2(S)) / ((l_2(Q_0)l_1(S)))$.

(Calculate the equations of the straight lines l_1 and l_2 arising in a doubling of T_1 .)

If the n th bit of l is one then

Set $T_1 = T_1 + P$ and set $f_1 = f_1(l_1(Q_0)l_2(S)) / ((l_2(Q_0)l_1(S)))$.

(Calculate the equations of the straight lines l_1 and l_2 arising in an addition of T_1 and P .)

Decrement n .

Step 4

Return f_1 .

Note that: l is the size of subgroup on which pairing is constructed. The divisor $(Q_0)-(S)$ is equivalent to the divisor $(Q)-(O_E)$. Since S was chosen randomly, it is likely that the points Q_0 and S in the support of $(Q_0)-(S)$ do not appear in any intermediate computations in the algorithm. Secondly, note that at each stage in the algorithm T_1 is the point obtained by computing $[m]P$ where m is the integer whose binary expansion is the top n bits of the binary expansion of l . The value f_1 is the evaluation at the divisor $(Q_0)-(S)$ of the function f defined such that $m((P)-(O_E)) = (T_1)-(O_E) + (f)$. Hence, at the end of the algorithm we have $T_1 = O_E$ and f_1 is the evaluation at $(Q_0)-(S)$ of the function g such that $l((P)-(O_E)) = (g)$.

2) Primitive Operations

As shown in the Miller algorithm, the major computations in pairing are computing and updating functions f and l . In this process, field operations are called. Among all the field operations, the *field multiplication (field squaring)* dominates the computation time. For example, one of the fastest pairing algorithms takes more than 3500 field multiplications, which is very large compared with other field operations.

During the pairing computation, the number of *field multiplication* operations can be derived by the following formula:

$$N_p = \sum_{L=1}^{h-1} (N_r \times (N_u \times N_{pu} + N_l \times N_{pl})) + N_l \times N_{pl}, \quad (1)$$

where N_p is the number of primitive operations, L is number of loops, h is the hamming weight of binary representation of the subgroup size, N_r is the number of repeats per loop, N_n is the number of function that update the value of f per repeat, N_{pu} is

the number of primitive operations per updating function, N_l is the number of l calculations, and finally N_{pl} is the number of primitive operations per l calculation. In order to reduce the computational overhead, we must select appropriate parameters, such that the subgroup size is a prime and the hamming weight is small.

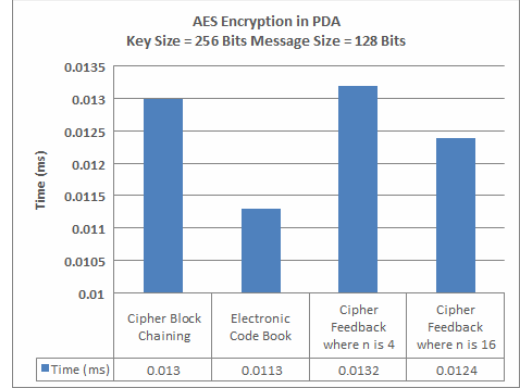


Figure 1 Time analysis of AES encryption for various block cipher modes.

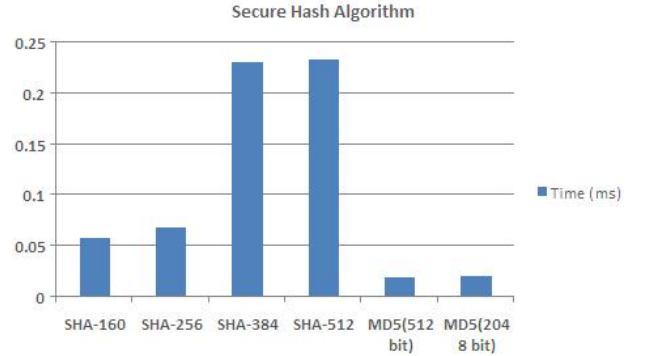


Figure 2 Time analysis of hash algorithms in Pocket PC.

IV. PERFORMANCE EVALUATIONS

In this section, we conduct benchmark studies to evaluate the performance of cryptographic algorithms such as AES, SHA, MD5, and Pairings in Pocket PC and wireless sensors. Our results are shown in follows.

A. Performance of Pocket PC

1) Performance of Advanced Encryption Standard in Pocket PC

Advanced Encryption Standard (AES) is a block cipher system that encrypts data in 128-bit blocks using a key of 128, 192 or 256 bits. We have used a key size of 256 bits and analyzed the time taken for AES encryption and decryption for various block cipher modes like Cipher Block Chaining (CBC), Electronic code Book (ECB), Cipher Feed-Back (CFB), and Output Feed-Back (OFB). The processor and memory specification of the Pocket PC is described in section Table 1. Note that AES encryption and decryption take almost the same time. In Figure 1, we present the performance of AES encryption for various block cipher mode in the Pocket PC.

2) Performance of hash algorithms in Pocket PC

We calculate the time taken for hash algorithms like Secure Hash Algorithm (SHA) and MD5 (Message Digest Algorithm 5) in Pocket PC and results are shown in Figure 2.

3) Performance of pairing in Pocket PC

Implementation of Pairing in PC is already done by Shamus Software Ltd [4]. We migrate the implementation of Pairing in Pocket PC using the Miracl library. The parameters used for our testing is given as follows: P is a 512 bit prime; the order of the group is $l = 2^{159} + 2^{17} + 1$; the curve: $y^2 = x^3 + x$ in Jacobian projective coordinate; and the embedding degree $k=2$. The field is on the F_p and F_{p^2} . Table 5 shows the performance of pairing in personal computer, Pocket PC and sensor.

Table 5 Time analysis of pairing in pc, Pocket PC and sensor (* derived from [4]).

Device	Processor /Microcontroller unit	Time in Milliseconds
PC*	1GHz Pentium III	20
Pocket PC	32 bit-624 MHz Intel Bulverde technology based RISC processor	550
Wireless Sensor	8 bit ATmega128L (8 such units)	10592.99
Wireless Sensor with optimized Field Multiplication	8 bit ATmega128L (8 such units)	1550.94

B. Performance of Wireless Sensor

1) Evaluation of time taken for pairing in sensors

We are aiming at implementing pairing in sensor. In section III, we identify the primitive operations involved in it. All the primitive operations are field arithmetic and it was found that Pairing involves 3554 Modulus multiplications (derived below) and one modulus exponentiation. We calculate the time taken for these primitive operations in sensor and used Equation (1) to calculate the time taken for one Pairing operation in sensor.

We have used secp160k1 as the system parameter for the primitive field operations specified by Certicom Research [7]. Note that all the field operation is on the field F_p where P is 160 bit prime. Table 6 shows the time taken by the primitive field operations in sensors.

There are two implementations of Filed multiplications in TinyECC. One is straight forward implementation of field multiplication and the other one is specially optimized for a set of elliptic curves with verifiable properties. The later implementation improves the performance of field multiplications on the F_p where p was chosen as pseudo-Mersenne primes to allow for optimized modular reduction. The prime numbers can be represented as $p=2^m-w$ where w is binary and $w < 2^m$. The $2m$ bit result R' can be split into two m -bit halves r'_1 and r'_2 and the reduction can be done based on the congruence $2^m \equiv w$. Comparing to the Montgomery

reduction, the reduction for pseudo-Mersenne primes requires substantially less effort on devices with small processor word sizes [2].

Table 6 Time analysis of primitive field operations of pairing in sensor.

Primitive pairing operation	Time in milliseconds
Normal Field multiplication	2.9595692952473955
Optimized Field multiplication	4.1537814670139 E -1
Field exponentiation	74.6811760796441

In order to compute how much time to compute pairings in wireless sensors, we utilize Equation (1) as follows. In our case, the $h=3$; $N_r=142, 17$; $N_u=1$; $N_{pu}=2$; $N_f=1$; $N_{pl}=20$ or 28 (point doubling and point addition). Thus, we can derive:

$$N_p = 142 \times (1 \times 2 + 1 \times 20) + 1 \times 28 + 17 \times (1 \times 2 + 1 \times 20) + 1 \times 28 = 3554.$$

Time taken for pairing can be computed as:

$$\text{Total time} = N_m * \text{time}_{mul} + N_e * \text{time}_{exp}, \quad (2)$$

where N is the counts of operations and time is the time consumption for one operation. The total time when using normal multiplication is given as follows:

$$3554 * 2.9595692952473955 + 74.6811760796441 = 10592.990451388887707 \text{ms.}$$

The total time using optimized multiplication is:

$$3554 * 0.4153781467013889 + 74.6811760796441 = 1550.9351094563802506 \text{ms.}$$

Based on our performance evaluation, we can see the Parings operations are very *heavy* for wireless sensors.

2) Evaluation of time taken for primitive ECC operation in sensors

We describe the ECC algorithms in section III and identify the primitive operations of ECC. Table 7 shows the time taken for these operations in the sensor. We use secp160k1 as the system parameters and all operations are performed on a 168-bit point. For point multiplication, the point is multiplied by a 168-bit number.

Table 7 Time analysis of primitive ECC operations in sensors

Primitive operations of ECC	Time in Seconds
Point Addition	0.1958179473876953
Point Doubling	0.180427205503646
Point Multiplication	10.722556644015842
Optimized Point Multiplication	7.449531555175781

3) Evaluation of time taken for Encryption and Hash Operations in sensor

We evaluate the performance of SHA-160 and Cipher Block Chaining (CBC) mode encryption in sensors. CBC encryption operates on a fixed size block of the input message and each plaintext block is XORed with the previous ciphertext block and then encrypted using the key. The first plaintext block is encrypted with a dummy block called initialization vector. We utilize the CBC code from TinySec [1], where the

block size and the initialization vector are 64 bits, and the encryption key is 256 bits. The evaluation results are shown in Figure 3.

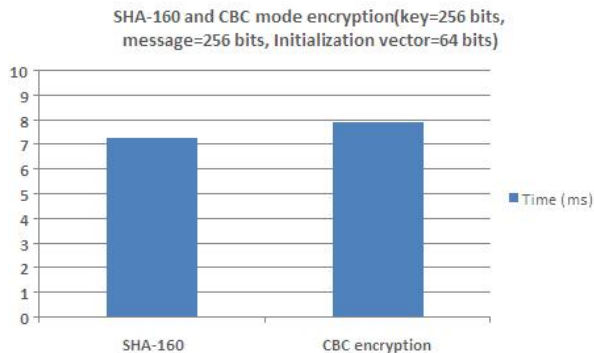


Figure 3 Time analysis of encryption and hash algorithm in Pocket PC.

C. Summary of the performance evaluation

Based on our conducted performance evaluations, we summarize our performance evaluations as follows. AES encryption in electronic code book mode takes much lesser time in Pocket PC compared to the other block cipher modes. Comparing the performance of SHA and MD5 in Pocket PC, we see that MD5 (512 bit) runs much faster than SHA-512. Our performance evaluation shows that we can make great improvements when we apply optimized algorithms; e.g., the computation time is reduced from about 10 seconds to 1.5 seconds.

Computation time improvements for pairing operations can be addressed based on (1) and (2).

In (1), we can do the followings to reduce the N_p :

- (a) Make hamming weight h as low as possible to reduce the number of loops. In our implementation, we take value $h=3$.
- (b) Reduce the value of N_r . Note that N_r is determined by l (size of subgroup) and the second bit 1 from the most significant bit of the subgroup size l . If $h=3$, the summation takes 2 turns and N_r is 142 and 17, respectively ($l=2^{159}+2^{17}+1=2^{17}(2^{142}+1)+1$). However, this is 160-bit subgroup incorporating 512-bit Prime P in the F_p . For sensor-level implementations, we can change p and l to smaller primes. If we can find a smaller l (128-bit) with small hamming weight ($h=3$) and the second bit 1 of l locates about the middle between the most significant bit and the least significant bit, we can significantly reduce the value of N_r to improve the performance of Pairings computation.

(c) Reduce the N_{pl} by mixed projective coordinates. Table 3 shows explicitly using mixed addition in Jacobian and Affine or Chudnovsky and Affine, only 8M and 3S are needed. N_{pl} can be reduced from 28 to 23 for point addition. In (2), we can reduce the $time_{mul}$ to improve the performance since field multiplications take up more than 99.2% of the total time. We can achieve this goal by adopting more efficient field multiplications. In [1], the researchers implemented pairings in smartcards and utilized a hybrid field multiplication approach

that uses the combination of row-wise and column-wise multiplication techniques. Their solution can reduce the number of memory accesses during the computation, thus achieve a high performance in the smartcard. The time of calculating pairing in ST22 processor@33Mhz is about 752 (ms), which is faster than our calculated value in sensors. The Micaz has 8 slower processor@7.37Mhz, and we can expect a better result in the sensors if we improve the field multiplication algorithms.

V. RELATED WORK

Among the related researchers, [7] gave the first implementations of Tate Pairings in Smartcard; the writers reported how the use of supersingular curves can be used in pairing implementation in embedded systems. In [2], the author have implemented and benchmarked operations in ECC with different system parameters on 8-bit CPU. Most of existing solutions to compute Parings are based on Miller algorithm [9]. However, no existing solutions evaluate Pairings operations for wireless sensors.

VI. CONCLUSION

In this paper, we evaluate the computational performance of cryptographic algorithms for Pocket PC and wireless sensors. Our study shows that purely software based solutions are suitable for Pocket PC; whereas wireless sensors have the difficulty to compute computational intensive operations, such as Parings.

In the future we aim at building an efficient pairing implementation in sensors by 1) applying efficient Pairings algorithms for sensors, such as [10]; 2) identifying the most efficient curve and system parameters for Parings in sensors; 3) specially optimizing the field operations for sensors.

REFERENCES

- [1] TinySec, <http://www.tinyos.net/tinyos-1.x/tos/lib/TinySec/>.
- [2] Nils Gura, Arun Patel, Arvinderpal Wander, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs", in *Workshop on Cryptographic Hardware and Embedded Systems* 2004.
- [3] D. Hankerson, A. Menezes, S. Vanstone "Guide to Elliptic Curve Cryptography", Springer-Verlag, 2004.
- [4] Miracl, by Shamus Software Ltd, <http://indigo.ie/~mscott/>
- [5] TinyEcc, by Cyber Defense Laboratory, NCSU, <http://discovery.csc.ncsu.edu/software/TinyECC/>
- [6] D. Boneh and M. Franklin. Identity-Based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
- [7] Certicom Research. Standards for efficient cryptography - SEC2: Recommended elliptic curve domain parameters. http://www.secg.org/download/aid-386/sec2_final.pdf, September 2000.
- [8] D. Huang, "Pseudonym-Based Cryptography for Anonymous Communications in Mobile Ad-hoc Networks", to appear in *Special Issue on Cryptography in Networks, International Journal of Security and Networks (IJSN)*, 2007.
- [9] V. Miller, "Short Programs for Functions on Curves," Unpublished manuscript, 1986.
- [10] I. Blake, K. Murty, and G. Xu, "Refinements of Miller's Algorithm for Computing Weil/Tate Pairing", *Journal of Algorithms*, Vo. 58, 2, Pages 134-149, 2006.