Elliptic curve cryptography-based access control in sensor networks

Haodong Wang, Bo Sheng and Qun Li*

Department of Computer Science,
College of William and Mary,
Williamsburg, VA, USA
E-mail: wanghd@cs.wm.edu
E-mail: shengbo@cs.wm.edu
E-mail: liqun@cs.wm.edu
*Corresponding author

Abstract: Access control in sensor networks is used to authorise and grant users the right to access the network and data collected by sensors. Different users have different access right due to the access restriction implicated by the data security and confidentiality. Even though symmetric-key scheme, which has been investigated extensively for sensor networks, can fulfil the requirement, public-key cryptography is more flexible and simple rendering a clean interface for the security component. Against the popular belief that a public key scheme is not practical for sensor networks, this paper describes a public-key implementation of access control in a sensor network. We detail the implementation of Elliptic Curve Cryptography (ECC) over primary field, a public-key cryptography scheme, on TelosB, which is the latest sensor network platform. We evaluate the performance of our implementation and compare with other implementations we have ported to TelosB.

Keywords: Wireless Sensor Networs (WSNs); public-key crytography; Elliptic Curve Crytography (ECC); access control.

Reference to this paper should be made as follows: Wang, H., Sheng, B. and Li, Q. (2006) 'Elliptic curve cryptography-based access control in sensor networks', *Int. J. Security and Networks*, Vol. 1, Nos. 3/4, pp.127–137.

Biographical notes: Haodong Wang is a current PhD candidate in Computer Scicence Department of the college of William and Mary. His research area is security and privacy in wireless sensor networks.

Bo Sheng is a current PhD candidate in Computer Scicence Department of the college of William and Mary. His research area is wireless sensor networks.

Qun Li received the PhD in computer science from Dartmouth College. He is an Assistant Professor in the Department of Computer Science at the College of William and Mary. His research interests include wireless networks, sensor networks, security and privacy.

1 Introduction

The access control component in sensor networks is responsible for authorising and granting users the right to access the network and data collected by sensors. A sensor network collects a variety of data shared by the users of the network. Due to privacy reason or data clearance, access restriction may be enforced for users with different access rights. For example, in a sensor network deployed on a battlefield, a high rank official may have more information access right than a soldier: a soldier is given the access to the data related to his task and a higher rank official necessarily requires information gathering for an overall manoeuvre. While much work on sensor network security has been focusing on key management methods for symmetric key scheme to support secure inter-sensor communication, little attention has been paid to access control in sensor networks.

Public-key cryptography has been used extensively in data encryption, digital signature, user authentication, etc. Compared with the popular symmetric key cryptography widely used in sensor network, public-key cryptography provides a more flexible and simple interface requiring no key predistribution, no pairwise key sharing, no complicated one-way key chain scheme. It is a popular belief, however, in the sensor network research community that public-key cryptography is not practical because the required computational intensity is not suitable for sensors with limited computation capability and energy budget. The nascent exploration seems to disabuse this misconception. The recent progress in 160-bit Elliptic Curve Cryptography (ECC) implementation on Atmel ATmega128, a CPU of 8 Hz and 8 bits, shows that an ECC point multiplication takes less than one second, which proves public-key cryptography is feasible for sensor network security related applications. This paper details our different implementations of ECC schemes on TelosB platform.

This paper describes our implementation of access control based on ECC over primary field on TelosB, a sensor network research platform. We give a framework for how to use ECC to grant user access right to collected data and show the experimental results. To compare with the similar implementations, we also port the other two implementations (NCSU (Liu and Ning, 2005) and Harvard (Malan et al., 2004)) to TelosB and measure the performance of each scheme, respectively. Our experiments demonstrate that our implementation outperforms the other two implementations. This work is especially important since TelosB mote platform is becoming a standard testbed for sensor network research: a public-key cryptography implementation and performance evaluation and comparison are conducive to the research progress of the community.

Our implementations are conducted on TelosB mote (TPR2400), which is the latest product in the Mote family designed by the University of California at Berkeley for experimentation in sensor network research. It is of the size of two AA batteries integrating USB programming capability, an IEEE 802.15.4 radio with integrated antenna, a low-power MCU with extended memory and an optional sensor suite (TPR2420). Its detailed features include: IEEE 802.15.4/ZigBee compliant RF transceiver, 2.4 to 2.4835 GHz (a globally compatible ISM band), 250 kbps data rate, integrated onboard antenna, 16 bit, 8 MHz TI MSP430 microcontroller with 10 kB RAM, low current consumption, 1 MB external flash for data logging, programming and data collection via USB, optional sensor suite including integrated light, temperature and humidity sensor (TPR2420), supported by Berkeley's TinyOS operating system (Tinyos, 2005) and the NesC programming language (Gay et al., 2003).

The rest of the paper is organised as follows. Section 3 gives an introduction to ECC. Section 4 describes the framework for access control in a sensor network. Section 5 shows implementation details of ECC on TelosB mote. Section 6 evaluates the performance of our implementation and compares with other implementations. Section 7 concludes the paper.

2 Related work

NIST and SECG have specified example elliptic curves domain parameters at required security levels (Certicom Research, 2000; National Institute of Standards Technology, 2000). As other implementations, we follow the recommended parameters in our implementation.

Gura et al. (2004) implemented ECC and RSA on 8-bit microcontroller and compared their performance. In their ECC implementation, the elliptic curves are defined over standardised prime integer field GF(p). Some optimisation techniques are applied for point multiplication, such as projective coordinates, Non-Adjacent Forms (NAFs) and curve-specific optimisation. Additionally, the paper focuses on the optimisation of modular multiplication of large integers, which is a critical operation for ECC. The authors compared row-wise and column-wise multiplications and further proposed a new hybrid strategy to improve the performance. The experiments showed that 160-bit ECC execution time was reduced to less than one second and much faster than RSA-1024 operations.

Shantz (2001) presented an efficient technique to calculate modular division, which is an important arithmetic operation in ECC and other cryptography system. The idea is to compute y/x in one operation, instead of the previous method which first computes $1/x$ and then multiply it with y. Thus, this scheme reduces one multiplication in the modular division operation. The new algorithm can be applied in both GF(p) and GF(2^m) fields.

Woodbury et al. (2000) introduced another ECC system over Optimal Extension Fields (OEFs) (Bailey and Paar, 1998) GF(p^m), where p is chosen as the form of $2^n \pm c$. The authors present the implementation of a specific 134-bit ECC in detail. The experiments indicate that point multiplication can be performed within 2 sec.

Cohen et al. (1998) analysed the impact of coordinates system in ECC implementation. The authors measured the performance of Point Addition (PADD) and Point Doubling (PDBL) of different coordinate systems and proposed a new modified Jacobian coordinates which achieves the fastest doubling operation. Moreover, they introduced a mixed coordinates system, which divides exponentiation into suboperations and chose the best coordinates representation for each suboperation. Thus, this scheme combines the advantages of different coordinates system and improves the computation time significantly.

In Hasegawa et al. (1998), an implementation of 160-bit ECC cryptographic library for Elliptic Curve Digital Signature Algorithm (ECDSA) over prime field GF(p) on a CISC Microcontroller (MC16) was given. Experiments obtain a speed of 150 ms for signature generation and 630 ms for verification.

EccM (Malan et al., 2004) is an ECC system implemented over binary field GF(2^p). The average time for public key generation is claimed as 34 sec. In Benenson et al. (2005), researchers design a user authentication protocol based on EccM library and implements it on TelosB mote. However, the verification takes several minutes. Sizzle (Gupta et al., 2005) is another application of ECC system developed recently. The standard internet security protocol(SSL) is efficiently implemented in sensor motes by using ECC.

3 ECC introduction

In this section, we briefly give a background introduction about ECC and corresponding elliptic curve Diffie-Hellman and Digital Signature Algorithm.

3.1 Elliptic curve cryptography

In recent years, ECC has attracted much attention as the security solutions for wireless networks due to the small key size and low computational overhead. For example, 160-bit ECC offers the comparable security to 1024-bit RSA. An elliptic curve over a finite field *GF* (a Galois Field of order q) is composed of a finite group of points (x_i, y_i), where integer coordinates x_i, y_i satisfy the long Weierstrass form:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (1)$$

and the coefficients a_i are elements in GF(q). Since the field $GF(q)$ (q is a prime) is generally used in cryptographic applications, (1) can be simplified to:

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

where $a, b \in GF(q)$.

The elliptic curve group operation is closed so that the addition of any two points is a point in the group. Given two points P and Q, with the coordinates (x_1, y_1), (x_2, y_2), respectively, the addition results in a point R on the curve with coordinate (x_3, y_3), where x_3 and y_3 satisfy

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3) \tag{3}$$

such that

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

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

where

$$L = (y_1 + y_2)/(x_1 + x_2) \tag{6}$$

If $x_1 = x_2$ (note $x_1 + x_2$ is 0), then R is defined as a point at infinity, O. O is an identity element of the group. Each element in the group has an inverse that satisfies $P + (-P) = O$ and $(-P) + P = O$. Also, $P + O = O + P = P$. If $P = Q$, then $R = P + P = 2P$, and coordinate (x_3, y_3) is derived by

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

$$y_3 = {x_1}^2 + (L + 1)x_3 \tag{8}$$

where

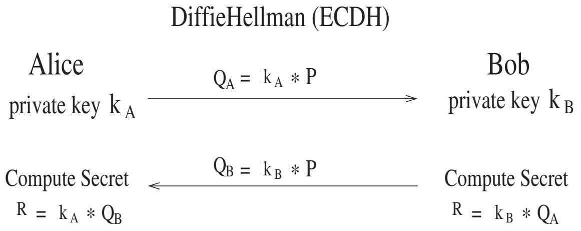
$$L = x_1 + y_1/x_1 \tag{9}$$

The ECC relies on the difficulty of the Elliptic Curve Discrete Logarithm Problem, that is, given points P and Q in the group, it is hard to find a number k such that $Q = kP$.

3.2 ECDH and ECDSA

The original Diffie-Hellman secret sharing protocol (Diffie and Hellman, 1976) requires a key of at least 1024 bits to achieve sufficient security. Unfortunately, low-power architecture, such as MSP430 and ATMega128, cannot afford the large memory overhead. Diffie–Hellman scheme based on ECC, however, can achieve the same security level with only 160 bit key size. A typical Elliptic Curve Diffie–Hellman (ECDH) scheme is shown in Figure 1. Initially, Alice and Bob agree on system base point P and generate their own public key Q_A and Q_B. To share a secret, Alice and Bob exchange their public keys and then use their own private key to multiply the other's public key. The result point R will be the secret. Eve, an eavesdropper, may overhear the communication and learn the public keys from Alice and Bob. However, with the knowledge of P, Q_A and Q_B, it is computationally intractable for Eve to get Alice and Bob's private keys. As a result, she can not figure out secret R.

Figure 1 An example of ECC version of Diffie-Hellman Protocol



ECC can also be used for Digital Signature Algorithm. Similarly, Alice and Bob have to agree on a particular curve with base point P. We assume the field is GF(p) and the order of P is q. When Alice sends a message to Bob, she attaches a digital signature (r, s) generated by following steps (suppose Alice has a private key x and a public key $Q = xP$):

1 choose a random key k in $[1, q - 1]$

2 compute kP, results a point with coordinate (x_1, y_1). Let $r = x_1$. Check r (mod q), go back to the first step if the result is zero

3 compute k^{-1} (mod q)

4 compute $s = k^{-1}(\text{Hash}(m) + xr)$, where Hash is an one-way hash function. Again, check s, go back to the first step if $s = 0$ and

5 (r, s) is the digital signature.

To verify the message m and the signature, Bob needs to do following steps.

1 compute $w = s^{-1} \bmod q$ and $H(m)$

2 compute $u_1 = H(m)w \bmod q$ and $u_2 = rw \bmod q$

3 compute $u_1P + u_2Q$, get the result point (x_2, y_2) and

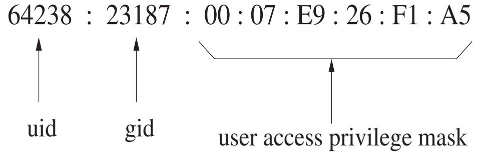
4 the signature is verified if $x_2 = r$.

4 Sensor network access control

We consider a large scale wireless sensor network deployed in a variety of environments, for example, at a hostile battlefield, in an office building or in a national park. The sensor nodes are battery-powered small devices which have very limited processing capabilities and memory space, such as the Berkeley MICA or MICA2 motes (Cressbow Technology INC, 2003). The sensor network is managed by a Key Distribution Center (KDC), which is responsible for generating all security primitives (i.e. random numbers, one-way hash function, Message Authentication Code (MAC), access list) and revoking users' access privilege if necessary.

Besides collecting and reporting data, sensor nodes are also capable of providing information services to users through the wireless channel. We assume the sensor nodes are densely deployed so that multiple sensor nodes can contact a user at the same time. To access the sensor network, users need to apply for the access permission from KDC. KDC maintains a user access list pool and associated user identifications. The access list defines the user's access privilege. A typical access list is composed of *uid*, *gid* and *user access privilege mask*. *uid* is a unique number to identify the user. *gid* is group identification. Multiple users who have similar task and access privilege can be organised into the same group. *user access privilege mask* is a number of binary bits, each bit represents a specific information or service. An access list example is shown in Figure 2. KDC issues a proper access list to each applicant. The information stored at the sensor nodes is divided into multiple access privilege levels. The user with a lower access privilege is not allowed to get the information that requires the higher privilege. Once a user passes the authentication check, the sensor nodes provide their local information to the user according to the provided access list.

Figure 2 An example of user access list. The access list is composed of three parts: *uid*, *gid* and access privilege mask. *uid* is a unique number assigned to each user. *gid* is a unique number assigned to the group to which the user belong access privilege mask is to define the user's access privilege to the system information


64238 : 23187 : 00 : 07 : E9 : 26 : F1 : A5
uid
gid
user access privilege mask

The essential part of access control is the authentication check for user's access list.

Initially, the KDC selects a particular elliptic curve over a finite field GF(p) (where p is a prime) and publishes a base point P with a large order q (where q is also a prime). The KDC picks a random number $x \in$ GF(p) as the system private key, and publishes its corresponding public key $Q = x \times P$. To access the sensor network, Alice comes to the KDC and gets her public key (Q_A), private key (q_A), and the certificate of her access list and public key ($T_A = C_A|\mathrm{ac}_A$ where '|' means concatenation). The following shows how to construct them in more detail. The KDC picks a random number $c_A \in$ GF(p) and then calculates Alice's public key constructor $C_A = c_A \times P$. Based on Alice's request and her background check, KDC issues a proper access control list ac_A and attaches it to public constructor C_A as the certificate for Alice's access list and public key. Meanwhile, a signature e_A is generated for the access list, where $e_A = H(T_A)$ (H is a $\{0,1\}^* \rightarrow \{0,1\}^q$ hash function). Then, the KDC constructs Alice's private key $q_A = e_A c_A + x$ and public key $Q_A = e_A \times C_A + Q$. Note q_A and Q_A satisfy $Q_A = q_A \times P$. Alice's access list T_A can be regarded as the certificate of her public key Q_A. Finally, Alice holds q_A, Q_A and T_A Figure 3 gives the definition of the notations.

Alice's access list authentication protocol is described in Figure 4. When Alice approaches a sensor node s_l, she sends her access request with access list T_A. Given access list T_A, s_l constructs Alice's public key $Q_A = e_A \times C_A + Q$. To verify Alice indeed holds private key q_A, node s_l uses the challenge as follows. s_l selects a random number $r \in GF(q)$ (to be used as the session key with Alice) and calculate its signature $H(r)$ over *mod*(q). Node s_l then generates temporary public key $Y_r = H(r) \times P$ and computes $Z_r = H(r) \times Q_A$. Then s_l encrypts the session key by doing $r \oplus X(Z_r)$, where $X(Z_r)$ is the X coordinate of point Z_r. Finally, s_l sends ciphertext $\langle z_r, Y_r \rangle$ to Alice, attached with a MAC of nonce N_A.

Figure 3 Notations

A :	User Alice
s_l :	a local sensor node
ac_A, T_A :	Alice's access list and certification
q_A, Q_A :	Alice's private and public key pair
P :	system elliptic curve base point
x, Q :	system private key and public key pair

With private key q_A, Alice can regenerate Z_r because $q_A \times Y_r = q_A \times H(r) \times P = H(r) \times Q_A = Z_r$. Alice then decrypts session key $r = z_r \oplus X(Z_r)$, and verifies if $Y_r = H(r) \times P$. If yes, Alice uses r as the session key to generate MAC for nonce N_A concatenated with her access privilege $\mathrm{ac_A}$, and sends to s_l.

Local sensor s_l decrypts the MAC message and verifies N_A and $\mathrm{ac_A}$. A successful verification proves that Alice is the owner of access list T_A. Finally, s_l replies the information requested by Alice, which again is encrypted by session key r.

Figure 4 Alice's access list authentication protocol

$$\begin{aligned}
Alice \rightarrow s_l &: T_A = (C_A|ac_A) \\
s_l \text{ computes } &: Q_A = e_A \times C_A + Q \\
&: \text{picks a random } r \in GF(q) \\
&: Z_r = H(r) \times Q_A, \\
&: Y_r = H(r) \times P, \\
&: z_r = r \oplus X(Z_r), \\
&: MAC(r, N_A). \\
s_l \rightarrow Alice &: z_r, Y_r, MAC(r, N_A) \\
Alice \text{ computes } &: q_A \times Y_r = q_A \times H(r) \times P = Z_r \\
&: X(Z_r) \oplus z_r = r \\
&: \text{decrypts } MAC(r, N_A) \\
Alice \rightarrow s_l &: MAC(r, N_A|ac_A) \\
s_l \rightarrow Alice &: MAC(r, reply)
\end{aligned}$$

A careful reader may find Alice has not verified local sensor s_l yet. As a result, an adversary may impersonate s_l and provide misleading information to Alice. A quick fix is to let s_l send its certificate to Alice and allow Alice to verify s_l in the same way. But it causes higher communication costs for local sensor node s_l. Obviously, the problem is beyond the access control covered in this paper because the fact whether Alice receives right answer does not harm the sensor network.

5 ECC implementation

We implement ECC cryptosystem on Telos-B mote powered by MSP430 microcontroller. The MSP430 incorporates an 8 MHz, 16-bit RISC CPU, 48 K bytes flash memory (ROM) and 10 K bytes RAM. This architecture provides 27 instructions and 7 addressing modes. The CPU also provides sixteen 16-bit registers. The first four are dedicated for special-purpose, such as programme counter, stack pointer and status register. The rest of the twelve are available for general use. Besides, the MSP430 also provides a peripheral hardware multiplier, which is capable of conducting up to 16 × 16 bits multiplication.

Given the limited processor resources, we concentrate most of our efforts on computation optimisation. The fundamental ECC operation is large integer arithmetic over either prime number finite field GF(p) or binary polynomial field GF(2^m) (where m is a prime). Because the two heavily used operations: multiplication and modular reduction, can be more effectively optimised if pseudo-Mersenne primes are picked up for elliptic curves compared with those of binary field (Gure et al., 2004), we limit our discussion in prime number finite field $GF(p)$ in this paper. Without further clarification, our following discussion is based on SECG recommended 160-bit elliptic curve: secp160r1.

5.1 Large integer operations

We implement a suite of large integer arithmetic operations, including addition, subtraction, shift, multiplication, division and modular reduction. Due to the space limit, we only present three of the most important functions: multiplication, division and modular reduction.

5.1.1 Multiplication

The efficiency of large integer multiplication dominates the overall performance of ECC operation. Gura et al. show that as much as 85% of execution time is spent on multiplication for a typical point multiplication in ECC. That means the optimisation on multiplication is critical for overall performance of our implementation. We have compared three different multiplication implementations (Gura et al., 2004; Liu and Ning, 2005; Malan et al., 2004), and finally decided to use Hybrid Multiplication proposed by Gura et al. (2004). To ease our explanation, we use three large integers as the examples for our following discussion: $A(a_{n-1}, a_{n-2}, \dots, a_1, a_0)$, $B(b_{n-1}, b_{n-2}, \dots, b_1, b_0)$, and $C(n_{2n-1}, c_{2n-2}, \dots, c_1, c_0)$, where $C = AB$. A and B both have length of n words, each word has k-bit size. The product C has $2n$ words.

The Hybrid multiplication is the combination of Row-wise multiplication and Column-wise multiplication. The Row-wise method fixes the multiplier b_i ($0 \le i \le n$) and multiplies it with every word of multiplicand A. Partial results are stored in $n + 1$ accumulator registers. Every time one row is finished, the last accumulator register can be stored in memory as part of the final results. On average, one memory load is required for each $k \times k$ multiplication. When integer size n is increased (integer size is 10 for curve secp160r1), the required number registers increase linearly in Row-wise method.

The Column-wise method, on the other side, computes the partial results of $a_i b_j$ (where $i + j = l$) for column l. After one column finishes, the last word of accumulator registers is stored as the part of final result. The Column-wise method only requires three accumulator registers and two more for operands. However, two memory load operations are required for each $k \times k$ multiplication. Considering a large number of data in ECC operations, unnecessary memory operations would lower the performance.

The Hybrid method takes advantage of Row-wise and Column-wise strategies. To optimise the memory operation, the Hybrid method merges a number (d) of columns together and then conducts Row-wise multiplication in each merged column. When d equals to 1, the Hybrid method becomes the Column-wise multiplication. When d equals to n, then it equals to Row-wise method. Therefore, a single memory load operation can be used for several multiplications. A larger d leads to fewer memory operations, but requires more registers. Since the MSP430 microcontroller only has 12 general registers, we can only implement the Hybrid method with column size $d = 2$, which requires 5 accumulator registers, 3 operand register and other 4 registers for pointer, temporary storage and loop control.

To achieve better performance and enable flexible control over registers, we implement the Hybrid multiplication in assembly language. Our experiments show that the performance of point multiplication improves about 5% with the Hybrid multiplication compared with the Column-wise method and improves another 5% with assembly language compared with original implementation with C.

5.1.2 Division

Modular division is another expensive operation in ECC. In Affine coordinate, each ECC operation of PADD and PDBL requires a modular inversion. The integer inversion is also required for ECC digital signature generation and verification. In our implementation, we adopt the Great Divide scheme proposed by Shantz (2001). We briefly explain the algorithm as follows.

Given a denominator x and numerator y, we want to compute the modular division y/x over $GF(p)$. This is equivalent to find r, so that

$$r \equiv \frac{y}{x}(\text{mod q}) \qquad (10)$$

To find r efficiently, we maintain following two invariant relationship:

$$Ay \equiv Ux \text{ and } By \equiv Vx \qquad (11)$$

where A, B, U and V are four auxiliary registers and assigned with initial values x, q, y and 0, respectively. Note the second invariant relationship is true even for $v = 0$ because algebraically the value of modulus is equivalent to zero in finite field. The division procedure repeatedly reduces the values of A and B in the following way. In each iteration, if either A or B is even, we divide by 2 both sides of the equation. If U or V is not even at that time, we can make it even by adding modulus q. If both A and B are odd, we add two equations together and then divide by 2 at both sides. By repeating this process, it is guaranteed that either value of

A or B reduces one bit in one iteration. The procedure stops when $A = B = 1$, the first equation becomes

$$y \equiv Ux \tag{12}$$

The value of U is our final result. If we initialise U with 1, this routine can be used to calculate an inversion of x. This algorithm works when x and q are relatively prime. Otherwise, the routine would return the greatest common divisor of A and B. The Great Divide finishes division or inversion operation in $2(log(x) - 1)$ steps.

5.1.3 Reduction

The modular reduction operation is as important as modular multiplication. Each multiplication must be followed by a reduction operation. Note the Great Divide algorithm does not work for modular reduction. Since we choose to use pseudo-Mersenne primes as specified in NIST/SECG curves, the modular reduction can be optimised by conducting a fixed number of integer additions. Because the optimisation is curve specific, we will explain in more detail in the section of ECC operation. Now, we discuss the modular reductions in digital signature generation and verification. In most cases, the order of an elliptic curve is not a pseudo-Mersenne prime, the optimisation cannot be applied for those reduction calculation. We choose the classic long division method to implement this operation. It may not be the most efficient algorithm, but it does not affect the overall performance much because very limited number of modular reductions are required in digital signature algorithm. We briefly describe the long division method as follows.

Given an integer x, we want to calculate

$$r \equiv x \bmod \mathrm{p} \tag{13}$$

where p is a prime.

1 Align the Most Significant Byte (MSB) of modulus p to the MSB of x, the lower bytes of p can be filled with zeros.

2 Starting with the MSB of x, divide the first two MSBs of x by the MSB of modulus q and get the quotient.

3 Multiply the quotient with the modulus and get a subproduct.

4 If the subproduct is greater than the remainder of x (over estimation), subtract the modulus from the subproduct.

5 Then subtract the subproduct from the remainder of x.

6 The procedure continues and goes back to step 2 if the MSB of the remainder becomes zero.

7 If the MSB of the remainder is not zero (under estimation), subtract the modulus from the remainder, and then go back to step 2.

8 The procedure stops when the remainder is less than modulus q.

The long division producer reduces the remainder of x by one byte in each iteration.

5.2 ECC Operations

In this section, we present our optimisation for ECC operation. We first discuss ECC PADD and doubling. We then introduce an optimized modular reduction for curve secp160r1. Finally, we explain several different optimisations for point multiplication.

5.2.1 ECC addition and doubling

The fundamental ECC operation is PADD and PDBL. The point multiplication can be decomposed to a series of addition and doubling operations. As discussed in previous section, PADD and PDBL in Affine coordinate require integer inversion, which is considered much slower than integer multiplication. Cohen et al. (1995) showed that these operations in Projective coordinate and Jacobian coordinate yield better performance. They further found addition and doubling in mixed coordinate, with the combination of Modified Jacobian coordinate and Affine coordinate, led to the best performance (Cohen et al., 1998). Consider an ECC point in Modified Jacobian coordinate, $P_1(X_1, Y_1, Z_1, aZ_1^4)$ and a point in Affine coordinate, $P_2(x_2, y_2)$, their addition results in the third point $P_3 = (X_3, Y_3, Z_3, aZ_3^4)$ in Modified Jacobian coordinate. The result is given by the following equations.

$$\begin{aligned} X_3 &= -H^3 - 2X_1H^2 + r^2 \\ Y_3 &= -Y_1H^3 + r(X_1H^2 - X_3) \\ Z_3 &= Z_1H \\ aZ_3^4 &= aZ_3^4 \end{aligned} \tag{14}$$

where $H = x_2Z_1^2 - X_1$ and $r = y_2Z_1^3 - Y_1$. The result of PDBL for $P_3 = 2P_1$ is given by the following formula.

$$\begin{aligned} X_3 &= T \\ Y_3 &= M(S - T) - U \\ Z_3 &= 2Y_1Z_1 \\ aZ_3 &= 2U(aZ_1^4) \end{aligned} \tag{15}$$

To estimate the computational complexity, we only consider large integer multiplication and squaring operations, and ignore those addition and subtraction since they are much faster. According to Equations (14) and (15), PADD requires 9 large integer multiplications and 5 squaring and point doubling requires 4 multiplications and 5 squaring.

The basic point operations can further be optimised for specific elliptic curves. In our case, the curve parameter a of secp160r1 equals to -3. For PDBL, M can further be reduced to

$$M = 3X_1^3 - 3Z_1^4 = 3(X_1 + Z_1^2)(X_1 - Z_1^2) \tag{16}$$

As a result, PDBL operation reduces to 4 multiplications and 4 squaring. Actually, aZ_3^4 does not have to be calculated in PADD, so the computational complexity reduces to 8 multiplications and 3 squaring. Our observation supports the choice of mixed coordinate, the performance of point multiplication improves around 6% compared with our previous implementation in Jacobian coordinate.

5.2.2 Modular reduction on ECC curve

Recall that modular reduction has to be applied after every large integer multiplication, it is also a performance critical operation. By taking advantage of pseudo-Mersenne primes specified in SECG curves, the complexity of the modular reduction operation can be reduced to a negligible amount. In this section, we use curve secp160r1 as an example to show how to do efficient reduction.

Suppose we use the 16-bit architecture, the multiplication result can be represented by

$$C(c_{19}, \ldots, c_{10}, c_9, \ldots, c_1, c_0)$$

where c_i ($0 \leq i \leq 19$) is a word with 16 bits and c_{19} is the most significant word. The 20-word integer can also be written as:

$$C = (c_{19}, \cdots, c_{10})2^{160} + (c_{10}, c_9, \cdots, c_1, c_0) \tag{17}$$

Given the field of curve secp160r1 $q = 2^{160} - 2^{31} - 1$, we can have $2^{160} \equiv 2^{31} + 1$. Therefore,

$$\begin{aligned} C &\equiv (c_{19}, \ldots, c_{10})(2^{31} + 1) + (c_{10}, c_9, \ldots, c_1, c_0) \\ &\equiv (c_{19}, \ldots, c_{10})2^{31} + (c_{19}, \ldots, c_{10}) \\ &\quad + (c_{10}, c_9, \ldots, c_1, c_0) \end{aligned} \tag{18}$$

Since each word has 16 bits, the first term in the result of Equation 18 can be further reduced to

$$\begin{aligned} &(c_{19}, \ldots, c_{10})2^{31} \\ &\equiv c_{19}2^{175} + c_{18}2^{159} + (c_{17}, \ldots, c_{10})2^{31} \\ &\equiv c_{19}2^{15}(2^{31+1} + (d_{15}, \ldots, d_1, d_0)2^{159} \\ &\quad + (c_{17}, \ldots, c_{10})2^{31} \\ &\equiv c_{19}2^{15} + c_{19}2^{46} + (d_0) * 2^{159} \\ &\quad + (d_{15}, \ldots, d_1)(2^{31} + 1) \\ &\quad + (c_{17}, \ldots, c_{10})2^{31} \\ &\equiv c_{19}2^{15} + c_{19}2^{46} + (d_0)2^{159} \\ &\quad + (d_{15}, \ldots, d_1)2^{31} + (d_{15}, \ldots, d_1) \\ &\quad + (c_{17}, \ldots, c_{10})2^{31} \end{aligned} \tag{19}$$

where $(d_{15}, \ldots, d_1, d_0)$ are 16 bits of c_{18}. Now, all terms in Equations (18) and (19) have at most 159 bit length, the reduction result is simply the addition of these terms.

5.2.3 Further optimisation

Examining the computational complexity, we notice that PADD is more expensive than PDBL. As we have discussed, point multiplication can be decomposed to a series of PADD and doubling, we would rather use more PDBL than point addition to compute the point multiplication. Morain et al. found NAFs is an effective way to achieve the lightest Hamming weight for scalar k in point multiplication kP, which results to use the least number of point additions to calculate kP (Morain and Olivos, 1990). For example, $255P$, or $(11111111)P$, requires 7 PADDs. But if we transform it to $(10000000 - 1)P$, which is $256 \times P - P$, only one addition is required. Note the point subtraction can be replaced by PADD because the inverse of an Affine point $P = (x, y)$ is $-P = (x, -y)$. We implement NAFs technique in Random Point Multiplication (RPM). According to our experiments, point multiplication with NAFs contributes at least 5% performance improvement.

Recall in the digital signature procedure in ECDSA, component r is generated by a point multiplication with the fixed base point of a selected elliptic curve. To further reduce the execution time, we precompute some partial results and apply the sliding window method (Kac, 1994) to speed up fixed point multiplication. Different from NAFs, sliding window scheme groups scalar k into a number of $s - bit$ bit-clusters, where s is also called window size. So, k can be represented by $k_m 2^{sm} + k_{m-1}2^{s(m-1)} + \cdots + k_0$, where k_i is a bit-cluster. If we precompute the point multiplication with every possible value of k_i, the number of point addition is bounded by $\lceil 160/s \rceil - 1$. Note the sliding window method does not reduce the number of PDBL operations. Obviously, this scheme requires extra memory space for storing partial results. In practice, we select window size $s = 4$. Correspondingly, there are 16 entries in the partial result table. Our experiments show that the sliding window method is more effective than NAFs for fixed point multiplication, the performance of sliding window method is more than 10% better than that of NAFs.

6 Experiments and performance evaluation

We have implemented the ECC security primitive and the proposed access control scheme on TelosB (TPR2420) motes, the latest research oriented mote developed by UC Berkeley. TelosB is powered by MSP430 microcontroller. MSP430 incorporates an 8 MHz, 16-bit RISC CPU, 48 K bytes flash memory (ROM) and 10K RAM. The RF transceiver on TelosB is IEEE 802.15.4/ZigBee compliant, and can have 250 kbps data rate.

In this Section, we first compare the performance of our implementation with other two related works: TinyEcc (Liu and Ning, 2005) and EccM (Malan et al., 2004). Then we present our ECC-based access control implementation and related experimental results. For comparison purpose, we also implement a symmetric-key based access control scheme and present the experimental result. We argue that the symmetric-key scheme suffers a number of problems even though it is computationally efficient for sensor nodes. Finally, we give an overall analysis to quantify the computation complexity.

6.1 Comparisons of the performance of ECC implementation

In experiments, we measure execution time, power consumption and code size of our implementation and compare the performance with other two released implementations, TinyEcc (Liu and Ning, 2005) and EccM

(Malan et al., 2004). We make appropriate modifications on their program to make them executable on TelosB platform. We choose secp160r1 as the elliptic curves parameters in all experiments.

We use the embedded system timer (32 kHz) to measure the execution time of major operations in ECC, such as point multiplication, point addition (PADD) and point doubling (PDBL). The energy consumption E can be calculated by $E = UIt$, where U is the voltage, I is the current and t is the time duration. TelosB motes are powered by two AA batteries, so U is approximated equal to 3.0 volts. The current value varies in different operations as given in Table 1 (abstracted from Moteiv Co. Telos datasheet. (2005)).

Table 1 The amount of current draw on different operations for TelosB motes

Operation	*Normal*	*Max*
MCU On, Radio Off	1.8 mA	2.4 mA
MCU On, Radio Rx	21.8 mA	23 mA
MCU On, Radio Tx	19.5 mA	21 mA

We first test point multiplication operation, which comprises PADD and PDBL. We consider two cases in point multiplication. One is multiplying large integer with a fixed point (base point) and the other one is with a random point. Fixed Point Multiplication (FPM) allows for optimisation by precomputing. We apply sliding window technique (Koc,1994) and set window size to 4, that is, precomputing $2^4 - 1 = 15$ points. In experiments, we randomly generate 20 large integers to multiply with the point and measure the average execution time.

Since ECC point multiplication consists of addition and doubling operations, we further evaluate these two operations individually. We generate random points and large integers for tests. Since a single operation takes very little time, to reduce the error of clock inaccuracy, we measure 100 operations every round and take the average value.

Table 2 gives the experimental results of execution time. PADD and PDBL of our implementation is superior to the other two implementations, which results in a faster point multiplication. Since EccM has no precomputation, there is no difference between the multiplications with base point and random point. Comparing with TinyEcc, we achieve about 4.8 sec improvement in FPM, and about 6.3 sec improvement in Random Point Multiplication (RPM).

Table 2 Execution time of point operations, including FPM, RPM, PADD and PDBL

	FPM	*RPM*	*PADD*	*PDBL*
Our codes	3.13 s	3.51 s	0.133 s	0.137 s
TinyEcc	7.98 s	9.86 s	0.315 s	0.300 s
EccM	88.43 s	88.43 s	0.262 s	0.255 s

Next, we implement ECDSA signature scheme and compare the performance with TinyEcc. The experimental results are given in Table 3. In fact, when signing a message, one FPM is the dominant operation. As we can see, the difference of signature time is very close to the difference of FPM. On the other hand, verification of ECDSA consists of one FPM and one random point multiplication. According to Table 2, we expect to achieve at least 11 sec improvement. However, TinyEcc implemented ECDSA in a different way with the assumption that the verifier is aware of the sender's public key. Thus, precomputation is conducted to optimise the performance for the RPM. We think this assumption is impractical for a scalable sensor network and we implement pure RPM in the verification process. That explains why we only gain less than 10 sec.

Table 3 Signature and verification execution time in ECDSA

	Signature	*Verification*
Our codes	3.35 s	6.78 s
TinyEcc	8.24 s	16.26 s

The power consumption is estimated by the formula $E = U\ It$. Thus, it is linear to the execution time. TelosB uses an ultra-low-power microcontroller MSP430. As given in Table 1, the current is 1.8 mA in active model. We use two new AA batteries, so the voltage is 3.0 V. Therefore, for our ECDSA implementation, generating a signature consumes roughly 18.09 mJ energy and verification costs 36.61 mJ. The whole authentication protocol consumes about 54.70 mJ for computation.

The following Table 4 compares the code size of three implementations. Our program uses 42.3 KB ROM and 1.6 KB RAM for ECDSA signature and verification protocol, where SHA-1, occupying more than 30 KB memory space, takes a large portion of codes. Compared with the ECDSA module in TinyECC, our implementation requires 3 KB more space in ROM and 0.5 KB more space in RAM. The reason is that we have implemented several optimisations in our assembly code, such as loop unrolling, to improve the performance. As the result, the code size is moderately inflated.

Table 4 Comparison of code size

	ECC library		*ECDSA*	
	ROM	*RAM*	*ROM*	*RAM*
Our codes	13.8 k	1.3 k	42.3 k	1.6 k
TinyEcc	12.5 k	1.3 k	39.2 k	2.1 k
EccM	17.6 k	1.1 k	–	–

6.2 *Performance of the access control protocol*

To simplify the experiments, we have implemented the user module on TelosB motes instead of PDAs. Considering the authentication procedure is communication intensive and does not require intensive computations, the experiment results capture the performance characteristics in reality even though TelosB motes (8 MHz) are much slower than normal PDAs (400 MHz).

Our implementation strictly follows the access control protocol presented in Section 4 except the data encryption/decryption part is not implemented due to the reason that TinySec (which provides block-cipher module) does not work with CC2420 radio module on TelosB. However, it does not affect our performance evaluation because encryption/decryption overhead is negligible (e.g. in RC5, the most expensive step (key setup) only costs 4 ms on ATmega128 (Ganesan et al., 2003)) compared with ECC exponentiation. The experiment is set-up as follows. Initially, the sensor mote is powered on and waiting for user's access request. The user's operations are divided into two stages. In the first stage, the user generates system parameters including system secret x and system public key Q, and then generates user's own access list, private key and certificate. This step basically simulates the operation of KDC. In the second stage, user communicates with the sensor and receives the information services. We start to count the time as soon as the sensor node receives the user's access request. We use challenge generation time and total transaction time as our performance indicators. The challenge generation time is user perceived delay from sending out the access request to receive the challenge from the sensor. The total transaction time is the amount of time for the sensor to approve the user, which also includes user's response time. We run the authentication for 20 times. Our experiment results show that challenge generation costs 10.12 s on the average with the standard deviation of 0.08 s and the overall access control time is 14.13 s on the average with the standard deviation of 0.09 s. The result of challenge generation time is consistent with our previous experimental results because the sensor has to do two RPM and one FPM. There is about 3.5 sec difference between the challenge generation time and the access control time. The reason is that we implement the user module on our sensor mode, and it takes around 3.5 sec for a sensor to compute a random point multiplication. In reality, this period of time should be much less because users are normally equipped with more powerful devices.

The code size for access control implementation is 47,106 bytes in ROM and 2064 bytes in RAM. The power consumption in TelosB combines the computational cost and communication cost. Using the same power consumption estimation as previous experiments, the computation consumes around 54.5 mJ. Since the current draw increases significantly when radio transceiver is on, as shown in Table 1, we calculate the communication cost as followings. Given 250 kbps radio transmission rate and 38 bytes in each packet, it takes one sensor node

$$38 \times 8(\text{bits})/250(\text{kbps}) = 1.2(\text{ms}) \qquad (20)$$

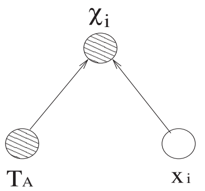
to transmit or receive a data packet. Without considering packet collision, the total transmission time is the product of Equation 20 with the number of packets. The energy cost for receiving a packet is $3.0(\text{v})21.8(\text{mA}) \times 1.2(rmms) = 78.5(\mu\text{J})$. Similarly, the energy cost for transmitting a packet is $3.0(\text{v})19.5(\text{mA})1.2(\text{ms}) = 70.2(\mu\text{J})$. During the authentication, it takes three packets for the user to send certification information to the sensor, four packets for the sensor to send the challenge to the user and the final packet for the user to respond the challenge. Totally, it costs $594.8(\mu\text{J})$ for the sensor in communication. Obviously, the computational cost dominates in the user authentication.

6.3 Comparison with symmetric key based authentication

To compare the performance of our ECC-based scheme with that of symmetric-key–based scheme, we also implement a symmetric key based access control scheme. The detail access control scheme is described as following.

Initially, each sensor node s_i is pre-loaded with a unique secret key x_i. When a user applies for access service at KDC, KDC issues a proper access list T_A to the user. At the same time, as shown in Figure 5, KDC also calculates and gives user the authentication code set $\{\chi_i\} = \{\text{Hash}(T_A||x_i)\}$, where $1 \leq i \leq n$. A simple symmetric-key based access control protocol is shown in Figure 6. Each χ_i is indexed by sensor id s_i. When the user approaches a local sensor, she first broadcasts an access request. An available local sensor replies with its sensor id s_i with a nonce N_B (N_B is a random number for the purpose of data freshness). The user looks up her table and picks the right χ_i as the secret key to encrypt her access list T_A, nonce N_B and a randomly generated nonce N_A (N_A is used to guarantee the data freshness). The user sends the encrypted message with a plaintext access list T_A to the local sensor. At the other side, the local sensor applies the hash function on the provided access list T_A with its own secret key x_i. If the result can successfully decrypt the ciphertext from the user (by verifying N_B), it is proved that the access list is genuine and issued by KDC. Otherwise, the user's request will be rejected. Finally, local sensor s_i replies the user with MAC(χ_i, N_A||data), where data is the user requested information. A successful decryption of N_A by the user automatically proves the sensor is genuine and can be trusted.

Figure 5 A simple and effective access list authentication by only one hash function: $\chi_i = H(T_A||x_i)$



Similarly, we implement both user and sensor modules on TelosB motes. We use 8-byte integer as user access list T_A, sensor secret x_i and authentication code χ_i. For one-way hash function, we choose 160-bit SHA-1. Since SHA-1 hash function outputs a 20 byte result, we only keep the lower 8 bytes as the result.

We perform the local authentication experiment 20 times. The average time duration for the sensor to authenticate the user is 75 ms. There are two rounds of communications, so the sensor spends 1.2ms × 2 in transmission and 1.2ms × 2 in receiving. Based on the time consumption and Table 1, we can calculate the power consumption each component as given in Table 5. When calculating the power consumption for communication, we choose the maximum current draw given in Table 5.

For the comparison purpose, we also list the performance of ECC-based scheme in Table. 5. Overall, it seems the symmetric-key based scheme is much more efficient than the ECC-based scheme. As we can see, our ECC-based scheme costs 130 times longer in authentication time and almost twice longer in communication delay. As a result, the total energy cost of our ECC-based scheme is almost 80 times more expensive than that of symmetric-key based scheme. However, the symmetric-key based scheme suffers a number of problems:

- *Scalability issue*: the symmetric-key based scheme does not scale. The reason is that users have to carry all the authentication codes. If the sensor nodes are pervasive in the environment, it will be a heavy burden for KDC to generate millions of authentication codes (one code for each sensor node) for users. Meanwhile, users may not afford the large storage space requirement.
- *Storage issue*: suppose each authentication code is 8 bytes. If there are 100,000 sensor nodes, any user has to store 800 MB data just for the authentication codes.
- *Re-deployment issue*: the symmetric-key scheme also has the problem in re-deployment. New sensors may be inserted to the existing network due to replacement for damages or network expansion. The problem comes when users try to access those new-deployed sensors because they do not have the authentication codes.
- *Key distribution issue*: normally, the symmetric-key based security schemes require a complicated key predistribution and a considerable memory space for storing predistributed keys. Many times, the key-distribution is difficult and problematic in pervasive computing environment, especially after the network has already been established.

Table 5 The performance comparison for the access control between symmetric-key based scheme and ECC-based scheme

	Sym. Scheme	ECC scheme
Auth. time	75.0 ms	10.1 s
Comp. cost	379.0 µ J	54.4 mJ
Comm. cost	316.8 µ J	594.8 µ J
Total cost	695.8 µ J	55.1 mJ

6.4 A performance anatomy of ECC point multiplication on TelosB

Since ECC point multiplication dominates the computational complexity in ECC operations, we are curious to analyse the performance anatomy in ECC point multiplication.

This analysis is based on 160-bit ECC curves. We use secp160r1 as the example. We also assume 4-bit sliding window method is used and partial results are precomputed. The computational cost for each window unit is 4 PDBL and 1 PADD. Given a 161 bit private key, there are 41 window units. Totally , 164 PDBL and 41 PADD are required to finish 1 point multiplication.

Figure 6 A simple and efficient symmetric-key based access control protocol

$$user \rightarrow s_i : \text{ access request}$$

$$s_i \rightarrow user : sid||N_B$$

$$user \rightarrow s_i : MAC(\chi_i, T_A||N_A||N_B)||T_A$$

$$s_i : \text{compute } \chi_i = H(T_A||x_i)$$

$$s_i : T_A||N_A||N_B = MAC^{-1}(\chi_i, T_A||N_A||N_B)$$

$$s_i \rightarrow user : MAC(\chi_i, N_A||data)$$

$$user : N_A||data = MAC^{-1}(\chi_i, N_A||data)$$

Large (160-bit) integer multiplication, squaring and reduction are the most expensive operations in PDBL and PADD. In the following analysis, we ignore the other integer operations. Each optimised point addition (in mixed coordinates) costs 8 large integer multiplications and 3 large integer squaring. Each optimised PDBL (in Jacobian coordinate) costs 4 large integer multiplications and 4 large integer squaring. In addition, each multiplication, squaring or shifting operation has to be followed by a modular reduction. Our program shows the PADD requires 12 modular reductions, and the PDBL requires 11 modular reductions. In total, each point multiplication costs $164 \times 4 + 41 \times 8 = 984$ large integer multiplications, $164 \times 4 + 41 \times 3 = 779$ large integer squaring and $164 \times 11 + 41 \times 12 = 2296$ large integer modular reductions. According to our tests, the cost of squaring is 90–95% of multiplication, and the optimised modular reduction costs about 25% of execution time of the multiplication. Converting squaring and reductions to multiplications by using above ratios, the total cost for one ECC point multiplication is equivalent to $984 + 779 \times 0.95 + 2296 \times 0.25 = 2298$ large integer multiplications. Our experiments show that the execution time of 1000 160×160 integer multiplications is 0.95 sec. We conclude multiplication, squaring and reduction operations in ECC point multiplication cost around 2.2 sec, which is roughly 73% of the total time (3.1 s) tested in our experiment. Based on the above analysis, we believe the performance of ECC operations on TelosB can be further improved by more refined and careful programming.

7 Conclusion

In this paper, we describe an ECC-based access control scheme in sensor networks. We give the protocol for the network to authorise a user to access the network and data collected by the sensors. We show our implementation of ECC on primary field on TelosB platform and compare the performance with other implementations that are ported to TelosB. Even though user access list authentication takes 10.1 sec, it is possible to further reduce the running time by using more refined and careful programming. Our experiment results demonstrate that public-key cryptography is feasible for sensor network security applications including access control. Our next step is to investigate more sophisticated access control schemes to alleviate the harm incurred by compromised sensor nodes.

Acknowledgment

We thank all the anonymous reviewers for their helpful comments. We thank Nils Gura for answering some of our questions. This project is partially supported by the US National Science Foundation under grant CCF-0514985.

References

Bailey, D.V. and Paar, C. (1998) 'Optimal extension fields for fast arithmetic in public-key algorithms', *Advances in Cryptography–CRYPTO'98*, pp.472–485.

Benenson, Z., Gedicke, N. and Raivio, O. (2005) 'Re-alizing robust user authentication in sensor networks', *Worshop on Real-World Wireless Sensor Networks*.

Certicom Research. (2000) 'SEC 2: Recommended elliptic curve domain parameters', *Standards for Efficient Cryptography Version 1.0*, September.

Cohen, H., Miyaji, A. and Ono, T. (1997) 'Efficient elliptic curve exponentiation', *Advances in Crytology-Proceedings of ICICS'97, Lecture Notes in Computer Science*, Springer-Verlag, pp.282–290.

Cohen, H., Miyaji, A. and Ono, T. (1998) 'Efficient elliptic curve exponentiation using mixed coordinates', *ASIACRYPT: Advances in Cryptology*.

Crossbow Technology INC. (2003) 'Wireless sensor networks', Available at: http://www.xbo.com/Products/Wireless_Sensor_Networks.htm.

Diffie, W. and Hellman, M.E. (1976) 'New directions in cryptography', *IEEE Transaction of Information and Theory*, Vol. IT-22, pp.644–654.

Ganesan, P., Venugopalan, R., Peddabachagari, P., Dean, A., Mueller, F. and Sichitiu, M. (2003) 'Analyzing and modeling encryption overhead for sensor network nodes', *WSNA03*, San Diego, CA, September.

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E. and Culler, D. (2003) 'The nesC language: a holistic approach to networked embedded systems', *Programming Language Design and Implementation (PLDI)*, June.

Gupta, V., Millard, M., Fung, S., Zhu, Y., Gura, N., Eberle, H. and Shantz, S.C. (2005) 'Sizzle: a standards-based end-to-end security architecture for the embedded internet', *Third IEEE International Conference on Pervasive Computing and Communication (PerCom 2005)*, Kauai, March.

Gura, N., Patel, A., Wander, A., Eberle, H. and Shantz, S.C. (2004) 'Comparing elliptic curve cryptography and rsa on 8-bit cpus', *CHES*, Boston, August.

Hasegawa, T., Nakajima, J. and Matsui, M. (1998) 'A practical implementation of elliptic curve cryptosystems over gf (p) on a 16-bit microcomputer', *Public Key Cryptography PKC'98*, pp.182–194.

Koc, C.K. (1994) 'High-speed RSA implementation', *RSA Laboratories TR201*, November.

Liu, A. and Ning, P. (2005) 'Tinyecc: elliptic curve cryptography for sensor networks', 15 September.

Malan, D.J. Welsh, M. and Smith, M.D. (2004) 'A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography', *In The First IEEE International Conference on Sensor and Ad Hoc Communications and Networks*, Santa Clara, CA, October.

Morain, F. and Olivos, J. (1990) 'Speeding up the computations on an elliptic curve using addition-subtraction chains', *Theoretical Informatics and Applications*, Vol. 24, pp.531–543.

Moteiv Co. Telos datasheet (2005) Available at: http://www.moteiv.com/products/docs/tmote-sky-datasheet. pdf.

National Institute of Standards and Technology (1999) 'Recommended elliptic curves for federal government use', August.

Shantz, S.C. (2001) 'From euclid's gcd to montgomery multiplication to the great divide', *Technical Report, Sun Microsystems Laboratories TR-2001-95*, June.

Tinyos (2005) Available at: http//www.tinyos.net/.

Woodbury, A.D., Bailey, D.V. and Paar, C. (2000) 'Elliptic curve cryptography on smart cards without coprocessors', *The Fourth Smart Card Research and Advanced Applications (CARDIS2000) Conference*, Bristol, UK, Septemper.