# Sensor Network Security:

# Elliptic Curve Cryptography on SunSPOTs

**Jens Mache, Samuel W. Bock, James Elwell,**
**Dennis P. Gosnell, Travis Mandel, Jonathan S. Perry-Houts**
Lewis & Clark College, Portland, Oregon/ USA
jmache@lclark.edu

**Abstract** - *Security is critical for wireless sensor networks. While several security mechanisms and protocols have been developed in the context of the Internet, many new challenges arise due to the characteristics of power-constrained wireless sensor networks. In this work, we focus on the Java-programmable SunSPOT device and elliptic curve cryptography (ECC). We present an analysis of run times and energy consumption. For our studies, we have developed our own implementation based on published pseudocode and Sun's BigInteger class. Our point multiplication on elliptic curves in prime finite fields is faster than that of Bouncy Castle, an open source ECC implementation. We investigate the impact of various parameters including key size, non-adjacent form, and prime versus binary fields. We believe such investigations to be an important step towards addressing the challenges of energy efficient security for Java-based sensor networks.*

**Keywords:** Wireless Sensor Networks, Security, Elliptic Curve Cryptography, SunSPOT, Performance Evaluation

## 1 Introduction

As wireless sensor networks become more common, security for these networks is imperative. This is especially true for devices such as medical sensors, where keeping information private is essential. These devices often transmit sensitive data that requires a cryptographic scheme that can provide confidentiality and integrity of data, and authenticity of those accessing the devices in the sensor network. Public-key cryptography provides all of these; however, the sensor devices have power constraints, in terms of both computation and battery power, that prevent the use of the most common public-key algorithm (RSA) because it is too computationally expensive. Elliptic Curve Cryptography (ECC) provides an alternative that grants comparable security strength with significantly less computation because it requires much smaller key sizes. In this paper, we implement ECC on SunSPOT devices from Sun Microsystems. We then time our implementation to compare it to similar implementations. We also compare the run time of ECC using a prime versus a binary finite field. To our knowledge, this is the first analysis of ECC on SunSPOTs.

In the next section we cover the background of our hardware and the fundamental mathematics we perform. In section Three we cover the related work. In section Four we describe our own implementation. We also found and solved a problem in a published version of the pseudocode for a particular algorithm used in ECC. In section Five we provide a performance evaluation comparing our timings to those of an open source version of ECC in Java. In section Six we summarize our results in our conclusion.

## 2 Background

In this background section we give more detail on wireless sensor networks, SunSPOTs, and ECC.

Wireless sensor networks are networks of miniaturized devices with integrated sensing, computing and communication capabilities. Devices coordinate amongst themselves to collaboratively detect events. Because of their small form factor, these devices can be embedded unobtrusively in the environment and enable up close monitoring of physical phenomena in places where computers would not usually be available and people would have trouble reaching. Embedded sensor networks can enable previously unprecedented sensing and control of the physical world across a broad spectrum of applications, ranging from traffic monitoring to nuclear reactor control.

In this paper we focus on Java-programmable sensor network devices called SunSPOTs. For more information on SunSPOTs see [7, 8, 15]. SunSPOTs can measure temperature, light, and acceleration with the default sensor board. They are programmed in Java Micro Edition (Java ME), which is very similar to the well-known standard edition of Java. The Squawk Virtual Machine runs directly on the hardware, without an operating system. SunSPOTs use the 802.15.4 IEEE standard for two-way wireless communication. The SPOTs are somewhat constrained in processing power and memory, however, as they have only a 180MHz CPU with 512KB of RAM and 4MB of Flash memory. They also have limited power, as they have only a single rechargeable battery [11].

The wireless communication and multi-node structure of a sensor network leave it vulnerable to external attacks. This was shown quite explicitly when a group at the University of Washington managed to use data sent by Nike+iPod pedometers to track people's movement over time [12]. They accomplished this using inexpensive, off the shelf electronics to create a grid of reading devices (also equipped with GPS) that intercepted the unique identifier sent wirelessly by the shoe sensor. This clearly demonstrated the danger of a lack of security for communication in sensor networks. However, the processing power, energy consumption, and memory of most sensors prevent the implementation of most security algorithms on these devices. For example, RSA encryption requires 1024-bit keys for sufficient security, which are too large to manipulate and store quickly and efficiently on sensor-class hardware.

This leads us to look for an encryption algorithm that can provide adequate security with much smaller key sizes. For this we turn to Elliptic Curve Cryptography (ECC) which, while mathematically more complicated than RSA, uses much smaller key sizes and is much faster to compute. A key of around 160-bits will provide adequate security today, whereas it takes a 1024-bit RSA key. Furthermore, the necessary ECC key size grows slower than the necessary RSA key size, as an ECC key 224-bits long provides similar security to a 2048-bit RSA key, which is the length that will be required for adequate security in the next few years [5].

For ECC, elliptic curves are defined over finite fields with either a binary or a prime characteristic (the size of the field). The binary finite field $F_{2^m}$ (where m is a large positive integer) and the prime finite field $F_p$ (where p is a large prime) use elliptic curves of the form:

$$y^2 + xy = x^3 + ax^2 + b$$

$$y^2 = x^3 + ax + b$$

respectively [6, 10, 14], where a and b are parameters that can be obtained from various organizations such as SECG [13]. The primary operation used in ECC is scalar point multiplication. The simplest way to do this is though multiple point additions, one of which is shown geometrically in Figure 1 [16]. Given points P and Q, their sum is the intersection of the line through P and Q and the elliptic curve (this line will intersect the curve at exactly one other point), the point R`. The inverse of R` is R, which is the sum of P and Q. We define the point at infinity(O), to be the sum of the points P and Q when the line drawn through them is vertical. In the case of scalar multiplication by k, a point is added to itself k times. When a point is doubled, P and Q in Figure 1 below would be the same point, and R` would be the intersection of the tangent line to that point with the elliptic curve. Because these operations are preformed in a finite field, we use modular arithmetic to ensure that all point multiples remain in the field. A small example with p = 23, a = 4, and b = 3 is shown in Figure 2 [3].

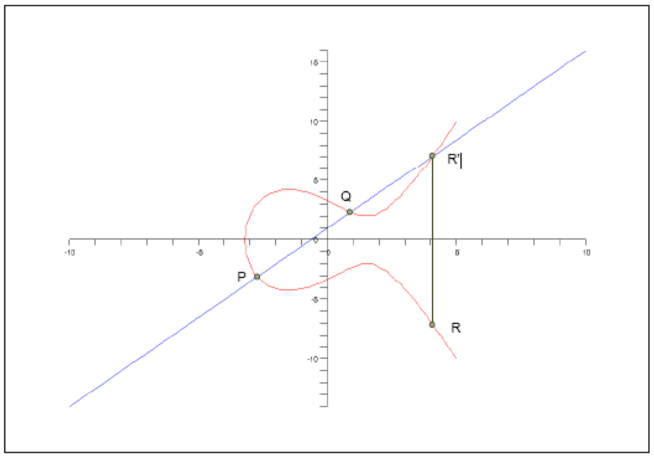


**Figure 1 : An elliptic curve over the reals with the addition of points P and Q.**

The security of ECC is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem. This problem entails the following: given points P and Q on an elliptic curve (where P is a base point, k is the private key -- a scalar-- and Q is the public key -- the product of k and P), solving for the scalar k in the equation (i.e. reversing our scalar multiplication):

$$Q = kP$$

This turns out to be very difficult to solve. For more on ECC, see [6, 9, 10, 14].

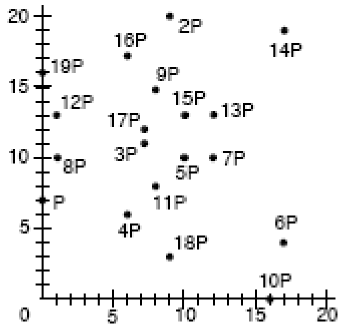


**Figure 2 : Scalar multiplication of point P. Note that only 19 multiples of P are shown, as 19P+P = O.**

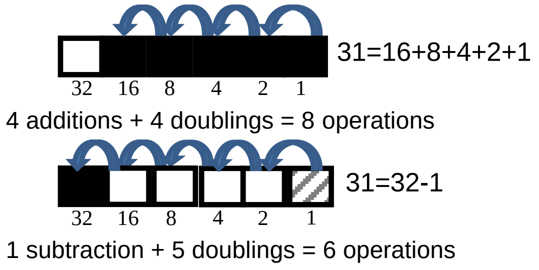
## 3 Related Work

There is a fair amount of related work dealing with ECC on various platforms. Java implementations have been used before in [1, 2, 10], although none of these implementations were on SunSPOTs and they used a variety of different algorithms. Edoh's [10] implementation was written in Java SE and tested on a Dell PC. Bernard's [1] was a platform-independent Java ME implementation, which he tested on a Nokia cell phone. Both of these include a significant amount of optimization and run significantly faster than many alternative implementations. Bouncy Castle has an open source Java implementation of ECC; however, it is too large to fit on SunSPOTs and much of the code must be cut out to reduce it to a usable size for these sensor devices. Bouncy Castle does not include a significant amount of optimization. There is an implementation of ECC in the "blue" version of the SunSPOT software development kit. This is implemented with the Java Cryptography Extensions. The JCA is a framework for accessing and developing cryptographic functionality for the not yet released Java platform. The implementations on MICAz and TelosB sensors by [17] focused on optimizing RSA and ECC for TinyOS, and found that both could be optimized to run on sensors, although RSA private key operations still take significantly longer than other operations. While these devices are different from SunSPOTs and these implementations are not in Java, the mathematics behind the implementations still applies. However, none of this related work provides a performance cross-comparison including the SunSPOT or between implementations using the two different finite field types (prime and binary) for ECC.

## 4 Implementation

In addition to deploying existing ECC implementations of Elliptic Curve Cryptography on the SunSPOTs, we worked to create and test our own. We used [6, 14] detailing the mathematical basis for ECC along with pseudocode and algorithm descriptions provided by Edoh and Bernard, and Java code used by other ECC libraries to program our own version.

To deal with the large numbers required by ECC we use a modified version of the Java SE BigInteger class, which is not normally available on Java ME platforms. The point addition code (the fundamental operation in ECC) we adapted from the algorithm used in [6, 10]. For scalar multiplication, the operation actually used in encryption and signing, we used the algorithm described by Simon Bernard in [1]. The scalar multiplication function uses the NAF (non-adjacent form) of k to perform point doublings whenever possible, and thereby minimize the number of point additions necessary. It does this by representing a number with -1, 0, and 1. Starting with the rightmost bit, it doubles the point for each bit, subtracts the point from the running total whenever it encounters a -1, and adds the point whenever it encounters a 1 (see Figure 3). This is important because point addition is an expensive operation when performing ECC, and using point doublings to eliminate additions can provide significant speedup. Additionally, we have written functions to deal with elliptic curves over both prime and binary fields, although only the prime field functions have been tested so far.



31=16+8+4+2+1
32 16 8 4 2 1
4 additions + 4 doublings = 8 operations
31=32-1
32 16 8 4 2 1
1 subtraction + 5 doublings = 6 operations

**Fig. 3: Example of computing 31P using the binary form (top) and the non-adjacent form (bottom). Each black box represents a 1, each white box represents a 0, and each striped box represents a -1. Note that the NAF method uses less operations than the binary method.**

### 4.1 Correction of Published Pseudocode

We discovered an omission in a published version [10] of the affine point addition algorithm. It is important to check whether the point P is its own inverse (which occurs when y =0) when doubling P. If this case is not checked and a point that falls into this type is the input, the algorithm may return a point that is off the elliptic curve, or may fail completely. The following improved version of the algorithm covers this case (the modification is denoted *):

```
Addition of points on $E(F_p)$ .
Input: Elliptic curve $E(F_p)$ with the points
$P_1$ $(x_1,y_1)$ and $P_2$ $(x_2,y_2)$
Output: $Q=P_1+P_2$.
if $P_1=O$ then return $Q \leftarrow P_2$
if $P_2=O$ then return $Q \leftarrow P_1$
if $x_1 = x_2$ then
        if $y_1 = y_2$ then
                *if $(y_1=0)$ return $(Q \leftarrow O)$
                else $\lambda \leftarrow (3x_1^2 + a)/(2y_1) \bmod p$
        else return $(Q \leftarrow O)$
else $\lambda \leftarrow (y_2 - y_1)/(x_2 - x_1) \bmod p$
$x_3 \leftarrow \lambda^2 - x_1 - x_2 \bmod p$
$y_3 \leftarrow \lambda(x_1 - x_3) - y_1 \bmod p$
return $Q \leftarrow (x_3,y_3)$.
```

This problem comes up quite frequently. For example, on the curve $y^2=x^3+4x+3$ with p = 23, the point (16,0) is on the curve since $x^3+4x+3=16^3+4(16)+3=4096+64+3=$

2+18+3=23=0 and $y^2=0^2=0$. The inverse of this point is (16,-1 *0) which is (16,0). Thus (16,0) + (16,0) is adding a point to its inverse, and the result should be the point at infinity. Simply doubling the point, as per the original algorithm, would result in trying to compute the inverse of zero when computing $\lambda$ , which results in undefined code behavior.

## 5 Performance Evaluation

The Bouncy Castle package [2] is offered in source under the GPL from bouncycastle.org and has already been implemented in JavaME. The Bouncy Castle crypto package contains Java code for many different cryptographic algorithms, but because we are focusing on ECC many of the included packages can be removed for the sake of saving space on the SPOTs' ROM. We did this by beginning with the main source file and successively adding in whatever necessary files the compiler reported missing.

The Bouncy Castle code includes all of the necessary classes to implement ECDSA (Elliptic Curve Digital Signature Algorithm, used to sign and verify messages) with ECC over an arbitrary field. The classes all run without modification on both the SPOTS and on a PC with any Java virtual machine installed. Run times for both message signing and signature verification return unexpected results regarding efficiency ratios between prime and binary fields. When run on a PC with an Intel dual core T7500 processor, message signing takes approximately twice as long with a binary field versus a prime field, and signature verification with a prime field runs in two-thirds the time of a binary field with comparable key length on average. However, on the SunSPOTs message signing consistently takes approximately three times longer in a prime field than in a binary field of comparable size and almost four times longer for signature verification in a prime field compared to a binary field. The timings (in seconds) comparing these two types are in the following table for signing and verifying a message:

**Table 1: Bouncy Castle timings on SunSPOTs in seconds.**

| Field Size | Signature | Verify |
|---|---|---|
| 191 bit Binary | 13.899 s | 17.283 s |
| 192 bit Prime | 38.905 s | 52.077 s |
| 239 bit Binary | 23.728 s | 30.563 s |
| 239 bit Prime | 66.273 s | 88.143 s |

However, one overall advantage of the prime field was that it was much easier to create test cases to test algorithm functionality.

We initially believed this reversal of efficiencies was most likely due to differences in memory access time and cache size between a PC with 4MB of L2 cache and a SPOT's ARM920T processor with 16KB. Upon further investigation, we discovered that Sun Microsystems has created two new assembly commands specifically for speeding up multiplication in binary fields [4]. On standard processors, prime field operations map well to standard integer operations, but binary fields do not. While we do not know for sure that these additional assembly instructions are the reason for the speedup on SunSPOTs, it seems to be a likely explanation. We have not yet investigated the specific function calls that are slowing down the operation or how they could be optimized for use on SunSPOTs and other small Java based devices, but it seems that a revision of the “BigInteger” class for dealing with large bit lengths would be beneficial. If it would be possible to reduce the memory footprint of the class that concatenates registers for larger bit lengths, it could drastically reduce the time spent on encryption.

Through this investigation of efficiencies it is possible for developers to decide whether ECC is feasible for their applications. Our goal was to make a new implementation that at least rivaled the speed of those that already existed, and in doing so determine if there are any modifications that could be made to increase the efficiency on a mobile sensor platform like the SunSPOTs.

We tested our algorithm using the following parameters for 160 and 192 bit ECC from [13]:

**Table 2: Field parameters.**

| | |
|---|---|
| Prime (160) | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC73 |
| a | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC70 |
| b | B4E134D3FB59Eb8BAB57274904664D5AF50388BA |
| Base point | (52DCB034293A117E1F4FF11B30F7199D3144CE6D, FEAFFEF2E331F296E071FA0DF9982CFEA7D43F2E) |
| k | B4A134D1FA591b8BA1A7274103664D51F50188BA |
| Prime (192) | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFEE37 |
| a | 000000000000000000000000000000000000000000000000 |
| b | 000000000000000000000000000000000000000000000003 |
| Base point | (DB4FF10EC057E9AE26B07D0280B7F4341DA5D1B1EAE06C7D ,9B2F2F6D9C5628A7844163D015BE86344082AA88D95E2F9D) |
| k | 3DF5BB7BF830F63C77667331106F9001B27D39941032F5E4 |

Based on our results, we managed to decrease the time (in seconds) needed for point multiplication with ECC over a prime field when compared to Bouncy Castle, as shown in the table below. The implementation by [1] is still faster than ours on SunSPOTs because of his inclusion of another optimization, Jacobian projective coordinates:

**Table 3: Timings for 160 and 191 bit point multiplications (in seconds).**

| | |
|---|---|
| Bouncy Castle (160) | 24.89 s |
| Our code (160) | 13.64 s |
| Bernard (160) | 3.25 s |
| Bouncy Castle (192) | 38.56 s |
| Our code (192) | 19.87 s |
| Bernard (192) | 4.55 s |

Running the 192-bit multiplication resulted in an average maximum of 110.6 mA of current drawn from the battery. A back-of-the-envelope calculation suggests that we could run this multiplication 667 times on the SPOT's 750mAh battery before its power is depleted. This represents the lower limit, as this was using the Bouncy Castle implementation that took just over 38 seconds. A faster implementation would yield many more multiplications on one battery.

## 6 Conclusions

After implementing and measuring elliptic-curve cryptography on the Java-programmable SunSPOT sensor network devices, our main conclusions are as follows:

1. Public-key cryptography on the Java-programmable SunSPOTs is becoming a viable option. 160-bit ECC point multiplication, which is as strong as 1024-bit RSA point multiplication, takes between 3.25 and 24.89 seconds on the SunSPOT device. As comparing the Bouncy Castle point multiplication times to the Bouncy Castle ECDSA times shows, the time it takes for a point multiplication is a good indication of how long it will take to actually encrypt and decrypt a message. The SunSPOTs power consumption of 110mA for 38 seconds running the Bouncy Castle code means that the battery (750mAh) would last for more than 600 digital signatures.

2. Whereas the Bouncy Castle [2] cryptographic library supports Java ME and thus works on the SunSPOT as-is, there seems to be room for improvement. When running prime fields on a cell phone, Bernard [1] improved on Bouncy Castle by up to a factor of 12. Running prime fields on SunSPOTs, we improved on Bouncy Castle's 192-bit point multiplication from 38.56 to 19.87 seconds, a speedup of about 2. Further improvements include Jacobian projective coordinates and pre-computation.

3. There is a flaw in Edoh's [10] pseudocode for point multiplication using affine coordinates. We corrected this omission, as explained in section 4.1.

4. Elliptic Curves over binary fields look even more promising than those over prime fields. As reported in Table 1, the run time of ECC on SunSPOTs is faster for fields using a binary characteristic instead of a prime characteristic.

Current and future work includes finishing our binary field implementation, further investigation into performance differences between binary and prime finite field characteristics, and further optimizations.

## Acknowledgements

This work is supported by NSF grants CNS-0720914, DMS-0649068, and by the John S. Rogers program. We would like to thank Liz Stanhope and Naiomi Cameron for helpful discussions, as well as Jeff Springer and William Sehorn.

## References

[1] Simon Bernard. "Efficient, Platform Independent Implementation of Elliptic Curve Cryptography for Mobile Devices"; Bachelor Thesis, Ruhr-University of Bochum. 2006.

[2] Bouncy Castle. http://www.bouncycastle.org

[3] Sheueling Chang, Hans Eberle, Vipul Gupta, Nils Gura. "Elliptic Curve Cryptography – How it Works"; Sun Microsystems Laboratories. 2004. http://research.sun.com/projects/crypto/

[4] Hans Eberle, Arvinderpal Wander, Nils Gura, Sheueling Chang-Shantz, Vipul Gupta, "Architectural Extensions for Elliptic Curve Cryptography over GF($2^m$) on 8-bit Microprocessors"; pp. 343-349, Proceedings of 2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05), 2005.

[5] Vipul Gupta. "Sizzle: SSL on Motes"; Winter 2005 CENTS Retreat, Jan. 2005. http://research.sun.com/projects/crypto/

[6] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. "Guide to Elliptic Curve Cryptography." Springer-Verlag. 2004.

[7] Jens Mache, Nirupama Bulusu, and Damon Tyman, "Sensor Network Lab Exercises Using Java and SunSPOTs"; poster at ACM SIGCSE, 2008.

[8] Jens Mache, Damon Tyman, and Nirupama Bulusu, "Making Sensor Networks Accessible to Undergraduates Through Activity-Based Laboratory Materials"; Proceedings of IEEE SECON, 2008.

[9] Kriangsiri Malasri and Lan Wang. "Addressing Security in Medical Sensor Networks"; Proceedings of the 1st ACM SIGMOBILE, 2007.

[10] Kossi Edoh. “Elliptic Curve Cryptography: Java Implementation”; Proceedings of the 1st annual conference on Information security curriculum development (InfoSecCD). 2004.

[11] “Project SunSPOT general FAQ” http://www.sunspotworld.com/docs/general-faq.php. June 2008.

[12] T. Scott Saponas, Jonathan Lester, Carl Hartung, Tadayoshi Kohno. “Devices That Tell On You: The Nike+iPod Sport Kit”; University of Washington, Department of Computer Science and Engineering, Technical Report, November 30, 2006.

[13] Certicom Research. “Standards for Efficient Cryptography Group: SEC 2: Recommended Elliptic Curve Domain Parameters”; Version 1.0, 2000.

[14] N. Smart. “Elliptic Curve Cryptography.” Handbook of Information Security (John Wiley & Sons, Inc.), Vol. 2, 558-574, 2006.

[15] Damon Tyman, Akshay Dua, Jens Mache, and Nirupama Bulusu, “Contour Tracking: A Comprehensive Student Project for Sensor Network Education”, demo at ACM SIGCOMM, 2008.

[16] Uhsadel, Leif, Axel Poschmann, and Christof Paar. “An Efficient General Purpose Elliptic Curve Cryptography Module for Ubiquitous Sensor Networks”; Proceedings of Software Performance Enhancement for Encryption and Decryption (SPEED), 2007.

[17] Haodong Wang and Qun Li. “Efficient Implementation of Public Key Cryptosystems on MICAz and TelosB Motes”; College of William and Mary, Technical Report WM-CS-2006-7, October, 2006.