Implementing Public-Key Infrastructure for Sensor Networks

DAVID J. MALAN, MATT WELSH, and MICHAEL D. SMITH
Harvard University

We present a critical evaluation of the first known implementation of elliptic curve cryptography over $\mathbb{F}_{2^p}$ for sensor networks based on the 8-bit, 7.3828-MHz MICA2 mote. We offer, along the way, a primer for those interested in the field of cryptography for sensor networks. We discuss, in particular, the decisions underlying our design and alternatives thereto. And we elaborate on the methodologies underlying our evaluation.

Through instrumentation of UC Berkeley's TinySec module, we argue that, although symmetric cryptography has been tractable in this domain for some time, there has remained a need, unfulfilled until recently, for an efficient, secure mechanism for distribution of secret keys among nodes. Although public-key infrastructure has been thought impractical, we show, through analysis of our original implementation for TinyOS of point multiplication on elliptic curves, that public-key infrastructure is indeed viable for TinySec keys' distribution, even on the MICA2. We demonstrate that public keys can be generated within 34 seconds and that shared secrets can be distributed among nodes in a sensor network within the same, using just over 1 kilobyte of SRAM and 34 kilobytes of ROM. We demonstrate that communication costs are minimal, with only 2 packets required for transmission of a public key among nodes. We make available all of our source code for other researchers to download and use. And we discuss recent results based on our work that corroborate and improve upon our conclusions.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Network operating systems*; C.4 [**Performance of Systems**]: Performance attributes; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*; D.4 [**Operating Systems**]: Security and Protection—*Cryptographic controls*; E.3 [**Data Encryption**]: Public key cryptosystems

General Terms: Algorithms, Design, Experimentation, Measurement, Performance, Security

Additional Key Words and Phrases: Diffie-Hellman, DLP, ECDLP, ECC, elliptic curve cryptography, MICA2, motes, sensor networks, TinyOS, TinySec

David J. Malan, Matt Welsh, and Michael D. Smith
Harvard University
School of Engineering and Applied Sciences
Cambridge, Massachusetts, USA
{malan,mdw,smith}@eecs.harvard.edu

This paper is an amplification of: *A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography*; David J. Malan, Matt Welsh, and Michael D. Smith; First IEEE International Conference on Sensor and Ad Hoc Communications and Networks; Santa Clara, California; October 2004.

1. INTRODUCTION

Wireless sensor networks have been proposed for such applications as habitat monitoring [Cerpa *et al.* 2001], structural health monitoring [Kottapalli *et al.* 2003], emergency medical care [Malan *et al.* 2004], and vehicular tracking [NEST Challenge Architecture 2002], all of which demand some combination of authentication, integrity, privacy, and security, implementation of which tends to require cryptographic primitives. Unfortunately, until recently, the state of the art in cryptography for sensor networks offered weak, if any, guarantees of these needs.

The limited resources boasted by today's sensor networks appear to render them ill-suited for the most straightforward implementations of security protocols. Consider the MICA2 mote [Crossbow Technology, Inc. 2004], designed by researchers at the University of California at Berkeley and fabricated by Crossbow Technology, Inc. Supported by Berkeley's TinyOS operating system [Hill *et al.* 2000] and the nesC programming language [Gay *et al.* 2003], this device offers an 8-bit, 7.3828-MHz ATmega 128L processor, 4 kilobytes (KB) of primary memory (SRAM), and 128 KB of program space (ROM). Such a device, given these resources, is seemingly unfit for computationally expensive or energy-intensive operations. For this reason has public-key cryptography often been ruled out for sensor networks as an infrastructure for authentication, integrity, privacy, and security [Karlof *et al.* 2004a; 2004b; Perrig *et al.* 2001; Perrig *et al.* 2004], even despite its allowance for secure rekeying of mobile devices.

But such conclusions have been backed too infrequently by actual data. In fact, prior to our original work [Malan 2004], little empirical research had been published to our knowledge on the viability of public-key infrastructure (PKI) for the MICA2, save for a cursory analysis of an implementation of RSA [Watro 2003].

Our work has aspired to fill this void. This paper in particular expands on our own prior work [Malan *et al.* 2004], providing not only a primer for those interested in cryptography for sensor networks but also additional details on our own experience with the same. Moreover, rather than present results in isolation, we expose in greater detail our motivation for various design decisions and elaborate on the methodologies underlying our evaluation of the same.

Through instrumentation of TinyOS, we first demonstrate that symmetric cryptography is tractable on the MICA2. By way of our own implementation of multiplication of points on elliptic curves, we then argue that PKI for secret keys' distribution is, in fact, tractable as well. We do not dwell in this work on tradeoffs one might need to make in order to integrate PKI with particular applications but, rather, on whether PKI is viable at all. With elliptic curves over $\mathbb{F}_{2^p}$, generation of public keys requires no more than 34 seconds (sec), and distribution of shared secrets requires no more than the same, using just over 1 KB of SRAM and 34 KB of ROM [Malan 2004; Malan *et al.* 2004]. With elliptic curves over $\mathbb{F}_p$ (and AVR assembly), the same operations can be implemented in far less time [Gura *et al.* 2004]. Communication costs, meanwhile, are minimal, with only 2 packets required for transmission of a public key among nodes.

To be sure, not all sensor networks (or applications thereof) require PKI, let alone any form of security. But with PKI comes capabilities that can certainly prove useful, among them the abilities to distribute symmetric keys securely and to sign messages digitally. Per Section 6, our original work on PKI for sensor networks [Malan 2004] is now part of a growing body of literature on the same.

We begin this paper's own look at PKI for sensor networks in Section 2 with an analysis of TinySec [Karlof *et al*. 2004b], TinyOS's existing symmetric infrastructure for the MICA2 based on SKIPJACK [National Institute of Standards and Technology 1988]. In Section 3, we address shortcomings in that infrastructure with a look at an implementation of Diffie-Hellman for the MICA2 based on the Discrete Logarithm Problem (DLP) and expose weaknesses in its design for sensor networks. In Section 4, we redress those weaknesses with our own implementation of Diffie-Hellman based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), the first such implementation to our knowledge [Malan 2004; Malan *et al*. 2004]. In Section 5, we discuss optimizations underlying our implementation. In Section 6, we discuss recent work by others that corroborates and improves upon our own findings. In Section 7, we propose directions for future work. In Section 8, we conclude.

Along the way, we offer a primer for those interested in this field of cryptography for sensor networks. In particular, we discuss the decisions underlying our design and alternatives thereto. We also elaborate on the methodologies underlying our evaluations of SKIPJACK and Diffie-Hellman for sensor networks. Toward this paper's end, we also provide a hyperlink to all of our source code for other researchers to download and use.

2. SKIPJACK AND THE MICA2

TinyOS offers the MICA2 access control, authentication, integrity, and confidentiality through TinySec, a link-layer security mechanism based on SKIPJACK in cipher-block chaining mode. An 80-bit symmetric cipher, SKIPJACK is the formerly classified algorithm behind the Clipper chip, approved by the National Institute for Standards and Technology (NIST) in 1994 for the Escrowed Encryption Standard [National Institute of Standards and Technology 1994]. TinySec supports message authentication and integrity with message authentication codes, confidentiality with encryption, and access control with shared, group keys. We evaluate in this section the performance of this mechanism, as the PKI we propose in later sections for symmetric keys' distribution will assume that we have access to efficient symmetric-key primitives.

The mechanism allows for an 80-bit key space, the benefit of which is that known attacks require as many 2^{79} operations on average (assuming SKIPJACK isn't reduced from 32 rounds [Biham *et al*. 1999]).[1] Moreover, as packets under TinySec include a 4-byte message authentication code (MAC), the probability of blind forgery is only 2^{-32}. This security comes at a cost of just five bytes (B): whereas transmission of some 29-byte plaintext and its cyclic redundancy check (CRC) requires a packet of 36 B, transmission of that plaintext's ciphertext and MAC under TinySec requires a packet of only 41 B, as the mechanism borrows TinyOS's fields for Group ID (TinyOS's weak, default mechanism for access control) and CRC for its MAC (Fig. 1).

[1] Although TinySec allows for 80-bit keys, its original implementation actually relied on 64-bit keys that were extended with 16 bits of padding.

Field	Length
Destination Address	2 B
Active Message Type	1 B
Group ID	1 B
Data Length	1 B
Data	29 B (max)
CRC	2 B
Total	36 B

(a)

Field	Length
Destination Address	2 B
Active Message Type	1 B
Data Length	1 B
Initialization Vector	4 B
Encrypted Data	29 B (max)
MAC	4 B
Total	41 B

(b)

Fig. 1. (a) TinyOS packet format without TinySec; (b) TinyOS packet format with TinySec.

2.1 Performance

The impact of TinySec on the MICA2's performance is reasonable. To assess TinySec's impact on packets' transmission time and round-trip time, we implemented BenchmarksM. A MICA2 running this TinyOS module forever transmits "pings" with 29-byte, random payloads to another mote running the same, who echoes the same in return. Each mote measures not only the time elapsed between `SendMsg.send(·,·,·)` and `SendMsg.sendDone()` but also that between `SendMsg.send(·,·,·)` and `ReceiveMsg.receive(·)`. All such measurements are logged to `TOS_UART_ADDR` for analysis by our implementation in Java of a `MessageListener` with which we determined the median, mean, standard deviation, and standard error for the MICA2's transmission and round-trip times, without and with TinySec, over 1,000 such measurements. A link to these tools' source code is offered toward this paper's end.

On first glance, it would appear that TinySec adds under 2 milliseconds (ms) to a packet's transmission time (Table I) and under 5 ms to a packet's round-trip time to and from some neighbor (Table II). However, the apparent overhead of TinySec, 1,244 microseconds (μsec) on average, as suggested by transmission times, is nearly subsumed by the data's root mean square (1,094 μsec). Round-trip times exhibit less variance, but more precise analysis of TinySec requires tighter benchmarks.

We thus instrumented TinyOS's TinySecM in order to measure the time required to invoke `encrypt()` and `computeMAC()` on 29-byte, random payloads, averaged over 1,000 trials. A link to our instrumentation's source code appears at this paper's end. Table III offers our results, which exhibit far less variance: encryption of a 29-byte, random payload requires 2,190 μsec on average, and computation of that payload's MAC requires 3,049 μsec on average. Overall, TinySec adds 5,239 $\pm$ 18 μsec to a packet's computational requirements. It appears, then, that some of those cycles can be subsumed by delays in scheduling and medium access, at least for applications not operating at full duty. Fig. 2, the results of an analysis of the MICA2's throughput, without and with TinySec enabled, puts the mechanism's computational overhead for such applications into perspective: on average, TinySec may lower throughput of acknowledged packets by only 0.28 packets per second. These results are in line with UC Berkeley's own evaluation of TinySec.[2]

[2]Per personal correspondence with Naveen Sastry, University of California at Berkeley.

	without TinySec	with TinySec
Median	72,904 μsec	74,367 μsec
Mean	74,844 μsec	76,088 μsec
Standard Deviation	24,248 μsec	24,645 μsec
Standard Error	767 μsec	779 μsec

Implied Overhead of TinySec	1,244 μsec
Root Mean Square	1,094 μsec

Table I. Transmission times required to transmit a 29-byte, random payload, averaged over 1,000 trials, without and with TinySec enabled. Transmission time is defined here as the time elapsed between `SendMsg.send(·,·,·)` and `SendMsg.sendDone()`. The implied overhead of TinySec on transmission time is given as the difference of the data's means. The root mean square is defined as $\sqrt{s_{\mathrm{w/o}}^2/1,000 + s_{\mathrm{w/}}^2/1,000}$, where $s_{\mathrm{w/o}}$ and $s_{\mathrm{w/}}$ are the data's standard deviations.

	without TinySec	with TinySec
Median	145,059 μsec	149,290 μsec
Mean	147,044 μsec	152,015 μsec
Standard Deviation	30,736 μsec	31,466 μsec
Standard Error	972 μsec	995 μsec

Implied Overhead of TinySec	4,971 μsec
Root Mean Square	1,391 μsec

Table II. Round-trip times required to transmit a 29-byte, random payload, without and with TinySec enabled, from one node to a neighbor and back again, averaged over 1,000 trials. More precisely, round-trip time is defined here as the time elapsed between `SendMsg.send(·,·,·)` and `ReceiveMsg.receive(·)`. The implied overhead of TinySec on round-trip time is given as the difference of the data's means. The root mean square is defined as $\sqrt{s_{\mathrm{w/o}}^2/1,000 + s_{\mathrm{w/}}^2/1,000}$, where $s_{\mathrm{w/o}}$ and $s_{\mathrm{w/}}$ are the data's standard deviations.

	`encrypt()`	`computeMAC()`
Median	2,189 μsec	3,038 μsec
Mean	2,190 μsec	3,049 μsec
Standard Deviation	3 μsec	281 μsec
Standard Error	0 μsec	9 μsec

Implied Overhead of TinySec	5,239 μsec
Root Mean Square	9 μsec

Table III. Times required to encrypt a 29-byte, random payload, and to compute that payload's MAC, averaged over 1,000 trials. The implied overhead of TinySec is given as the sum of the data's means. The root mean square is defined as $\sqrt{s_{\mathrm{w/o}}^2/1,000 + s_{\mathrm{w/}}^2/1,000}$, where $s_{\mathrm{w/o}}$ and $s_{\mathrm{w/}}$ are the data's standard deviations.

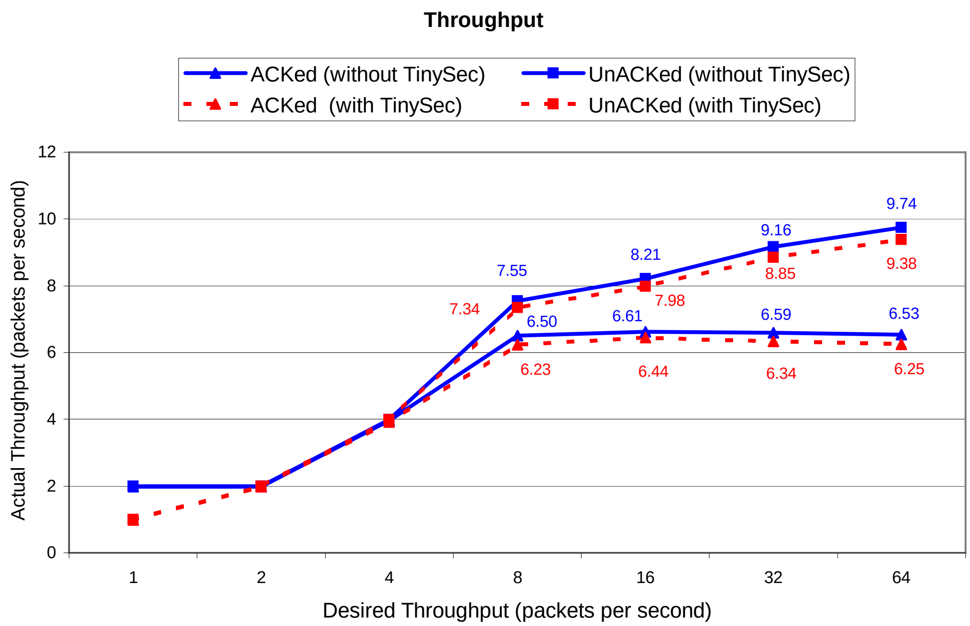

Fig. 2. Actual throughput versus desired throughput for acknowledged (ACKed) and unacknowledged (unACKed) transmissions between a sender and a receiver, averaged over ten minutes of transmission per level of desired throughput, where desired throughput is the rate at which calls to `SendMsg.send(·,·,·)` were scheduled by `Timer.start(·,·)`. ACKed actual throughput is the rate at which 29-byte, random payloads from a sender were received and subsequently acknowledged by an otherwise passive recipient. UnACKed actual throughput is the rate at which the sender actually sent such packets, acknowledged or not (*i.e.*, the rate at which calls to `SendMsg.send(·,·,·)` were actually processed). For clarity, where ACKed and unACKed throughput begins to diverge are points labelled with values for actual throughput. In environments with less contention for medium access than in ours, higher throughput is possible, without and with TinySec enabled.

2.2 Memory

Of course, TinySec's encryption and authentication does come at an additional cost in memory. To measure this cost, we utilized John Regehr's stacktool [Regehr 2004] to determine a TinyOS module's memory usage without and with TinySec enabled. Per Table IV, TinySec adds 454 B to an application's `.bss` segment, 276 B to an application's `.data` segment, 7,076 B to an application's `.text` segment, and 92 B to an application's maximal stack size during execution. For applications that don't require the entirety of the MICA2's 128 KB of program memory and 4 KB of primary memory, then, TinySec is a viable addition.

	without TinySec	with TinySec	Difference
`.bss`	384 B	838	454 B
`.data`	4 B	280 B	276 B
`.text`	9,220 B	16,296 B	7,076 B
stack	105 B	197 B	92 B

Table IV. Memory overhead of TinySec, determined through instrumentation of CntToRfm, an application that simply broadcasts a counter's values over the MICA2's radio. The `.bss` and `.data` segments consume SRAM while the `.text` segment consumes ROM. Stack is defined here as the maximum of the application's stack size during execution.

2.3 Security

As with any cipher based only on shared secrets, TinySec is, of course, vulnerable to various attacks. After all, the MICA2 is intended for deployment in sensor networks. For reasons of cost and logistics, long-term, physical security of the devices is unlikely. Compromise of the network, therefore, reduces to compromise of any one node, unless, for instance, rekeying is possible. Pairwise keys among n nodes would certainly provide some defense against compromises of individual nodes. But n^2 80-bit keys would more than exhaust a node's SRAM for n as small as 20. A more sparing use of secret keys is in order, but secure, dynamic establishment of those keys, particularly for networks in which the positions of sensors may be transient, requires a chain or infrastructure of trust. In fact, the very design of TinySec requires as much for rekeying as well. Though TinySec's 4-byte initialization vector (IV) allows for secure transmission of some message as many as 2^{32} times, that bound may be insufficient for embedded networks whose lifespans demand longer lasting security.[3] Needless to say, TinySec's reliance on a single secret key prohibits the mechanism from securely rekeying itself.

Fortunately, these problems of secret keys' distribution are redressed by public-key infrastructure. The sections that follow thus explore options for that infrastructure's design and implementation on the MICA2.

3. DLP AND THE MICA2

With the utility of SKIPJACK-based TinySec thus motivated and the mechanism's costs exposed, we next examine DLP, on which Diffie-Hellman [Diffie and Hellman 1976] is based, as an answer to the MICA2's problems of secret keys' distribution. DLP typically involves recovery of $x \in \mathbb{Z}_p$, given p, g, and $g^x \pmod{p}$, where p is a prime integer, and g is a generator of $\mathbb{Z}_p$. By leveraging the presumed difficultly of DLP, Diffie-Hellman allows two parties to agree, without prior arrangement, upon a shared secret, even in the midst of eavesdroppers, with perfect forward secrecy, as depicted in Fig. 3. Authenticated exchanges are possible with STS [Diffie *et al.* 1992], a variant of Diffie-Hellman.

With a form of Diffie-Hellman, then, could two nodes thus establish a shared secret for use as TinySec's key. At issue, though, is the cost of such establishment on the MICA2. Inasmuch as the goal at hand is distribution of 80-bit TinySec keys,

[3] To allow for secure transmission of as many as 2^{32} packets, it is actually necessary to modify TinySec so that it no longer writes a mote's address into the third and fourth bytes of a mote's IV.

any mechanism of exchange should provide at least as much security. According to NIST [National Institute of Standards and Technology 2003], then, the MICA2's implementation of Diffie-Hellman should employ a modulus, p, of at least 1,024 bits and an exponent (*i.e.*, private key), x, of at least 160 bits (Table V).

Unfortunately, on an 8-bit architecture, computations with 160-bit and 1,024-bit values are not inexpensive. However, modular exponentiation is not intractable on the MICA2. Fig. 4 offers the results of our instrumentation of one implementation of Diffie-Hellman for the MICA2 [BBN Technologies 2003]: computation of $2^x \pmod p$, where x is a pseudorandomly generated 160-bit integer and p is a 768-bit prime requires 31.0 sec on average; computation of the same, where p is a 1,024-bit prime, requires 54.9 sec. Assuming (generously) that nodes sharing some key need only be rekeyed every 2^{32} packets (at which time four-byte IVs are exhausted), this computation and that for $y^x \pmod p$, where y is another node's public key, seem reasonable costs for an application's longevity. Table VI details these operations' memory usage, which we measured with stacktool [Regehr 2004].

Of course, these measurements assume operation at full duty cycle, the energy requirements of which may be unacceptable, as the MICA2's lifetime decreases to just a few days at maximal duty cycle. To measure the cost in energy of modular exponentiation on the MICA2, we further instrumented the same implementation of Diffie-Hellman to raise and lower a general-purpose I/O pin at the start and end, respectively, of the primitive's execution. We then measured both time and average power consumption using an oscilloscope in order to calculate, using the frequency of the MICA2's clock, the primitive's total energy consumption [Shnayder *et al.* 2004].

Table VII reveals the MICA2's energy consumption for modular exponentiation: computation of $2^x \pmod p$ appears to require 1.185 J. Roughly speaking, a mote could devote its lifetime to 51,945 such computations.[4]

Of course, the implementation could be tuned for better performance. However, its computations ultimately require not only time but also memory. Mere storage of a public key requires as many bits as is the modulus in use. Accordingly, n 1,024-bit keys would more than exhaust a node's SRAM for n as small as 32. Although a node is unlikely to have—or, at least, need—so many neighbors or certificate authorities for whom it needs public keys, Diffie-Hellman's relatively large key sizes are unfortunate in the MICA2's resource-constrained environment. A key of this size would not even fit in a pair of TinyOS packets.

4. ECDLP AND THE MICA2

With ECC, secure distribution of 80-bit TinySec keys is possible using public keys with fewer bits than 1,024: 163 bits are sufficient (Fig. 5). Indeed, elliptic curves are believed to offer security computationally equivalent to that of Diffie-Hellman based on DLP with remarkably smaller key sizes insofar as subexponential algorithms

[4]For instance, Energizer No. E91, an AA battery, offers an average capacity of 2,850 mAh [Everyready Battery Company 2004]; it follows that no more than $2 \times 2{,}850$ mAh $\times$ 3600 sec/h $\div$ (7.3 mA $\times$ 54.1144 sec) $\approx$ 51,945 modular exponentiations would be possible with two AA batteries on the MICA2. Of course, this bound is generous, as the MICA2 effectively dies once voltage drops below 2 volts.

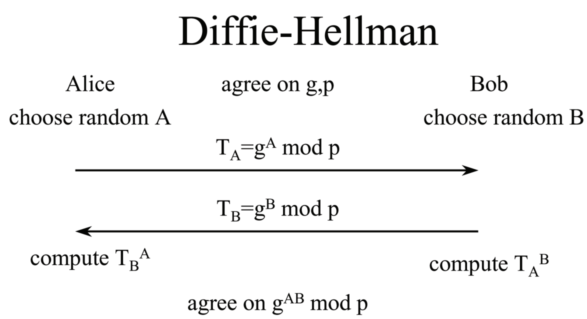

Fig. 3. Typical exchange of a shared secret under Diffie-Hellman based on DLP [Perlman 2003].

Bits of Security	Modulus	Exponent
80 b	1,024 b	160 b
112 b	2,048 b	224 b
128 b	3,072 b	256 b
192 b	7,680 b	384 b
256 b	15,360 b	512 b

Table V. Strength in bits (b) of Diffie-Hellman based on DLP for moduli and exponents of various sizes. "An algorithm that has a 'Y' bit key, but whose strength is equivalent to an 'X' bit key of such a symmetric algorithm is said to provide 'X bits of security' or to provide 'X-bits of strength'. An algorithm that provides X bits of strength would, on average, take $2^{X-1}T$ to attack, where T is the amount of time that is required to perform one encryption of a plaintext value and comparison of the result against the corresponding ciphertext value." [National Institute of Standards and Technology 2003]

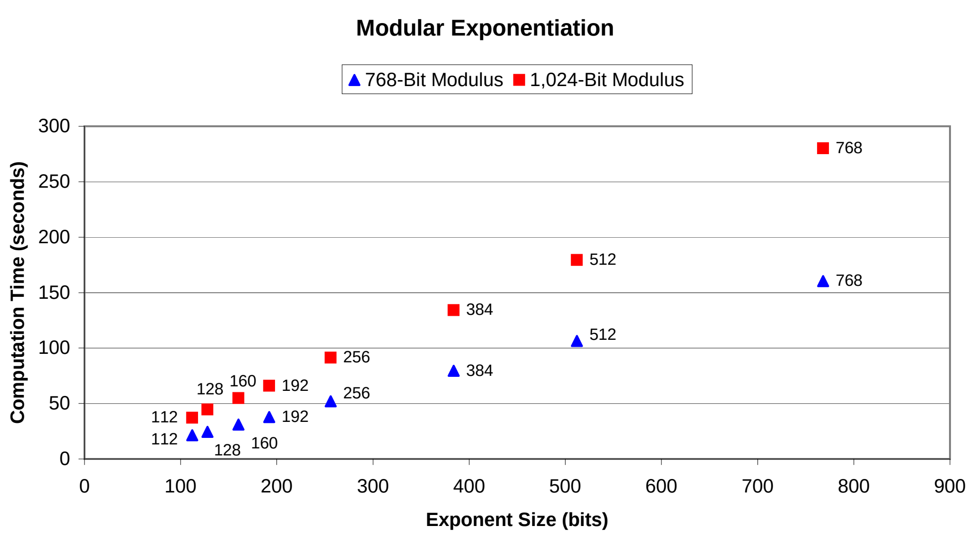

Fig. 4. Time required to compute $2^x \pmod{p}$, where p is prime, on the MICA2.

	768-Bit Modulus	1,024-Bit Modulus
`.bss`	852 B	980 B
`.data`	102 B	134 B
`.text`	11,334 B	11,350 B
stack	136 B	136 B

Table VI. Memory overhead of modular exponentiation, determined through our instrumentation of an implementation of Diffie-Hellman based on DLP on the MICA2 that computes $2^x \pmod p$, where x is a 512-bit integer and p is prime [BBN Technologies 2003]. The `.bss` and `.data` segments consume SRAM while the `.text` segment consumes ROM. Stack is defined here as the maximum of the application's stack size during execution.

	1,024-Bit Modulus, 160-Bit Exponent
Total Time	54.1144 sec
Total CPU Utilization	3.9897×10^8 cycles
Total Energy	1.185 Joules

Table VII. Energy consumption of modular exponentiation, determined through our instrumentation of an implementation of Diffie-Hellman based on DLP on the MICA2 that computes $2^x \pmod p$, where x is a 160-bit integer and p is a 1,024-bit prime [BBN Technologies 2003].

Size of SKIPJACK Key	Recommended Size of ECC Private Key	
	over $\mathbb{F}_p$	over $\mathbb{F}_{2^p}$
80 b	192 b	163 b
112 b	224 b	233 b
128 b	256 b	283 b
192 b	384 b	409 b
256 b	521 b	571 b

Fig. 5. Sizes in bits (b) of private keys necessary to exchange SKIPJACK keys securely using ECC over two different fields [National Institute of Standards and Technology 1999]. With ECC over $\mathbb{F}_{2^p}$, 163-bit keys are sufficient for the secure exchange of 80-bit SKIPJACK keys.

exist for DLP [Adleman 1979; Gordon 1993; LaMacchia and Odlyzko 1991; Rabin 1979], but no such algorithm is known or thought to exist for ECDLP over certain fields [Certicom Corporation 2000; Gaudry *et al.* 2000].

Elliptic curves offer an alternative foundation for the exchange of shared secrets among eavesdroppers with perfect forward secrecy, as depicted in Fig. 6. ECDLP, on which ECC [Koblitz 1987; Miller 1986a] is based, typically involves recovery over some Galois (*i.e.*, finite) field, $\mathbb{F}$, of $k \in \mathbb{F}$, given (at least) $k \cdot G$, G, and E, where G is a point on an elliptic curve, E, a smooth curve of the long Weierstrass form

$$y^2 + a_1xy + a_3y \equiv x^3 + a_2x^2 + a_4x + a_6, \qquad (1)$$

where $a_i \in \mathbb{F}$. Of recent interest to cryptographers are such curves over $\mathbb{F}_p$ and $\mathbb{F}_{2^p}$ (Fig. 7), where p is prime, as neither appears vulnerable to subexponential attack [Gaudry *et al.* 2000]. Though once popular, extension fields of composite degree over $\mathbb{F}_2$ are vulnerable by reduction with Weil descent [Frey and Gangl 1998]

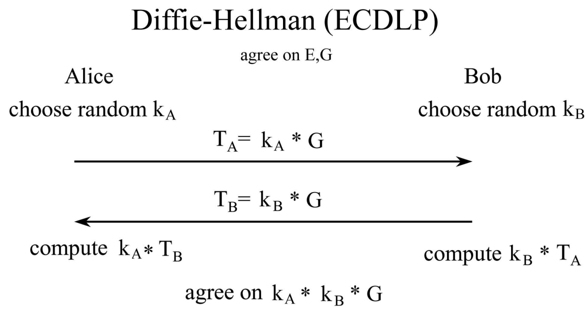

Fig. 6. Typical exchange of a shared secret under Diffie-Hellman based on ECDLP.

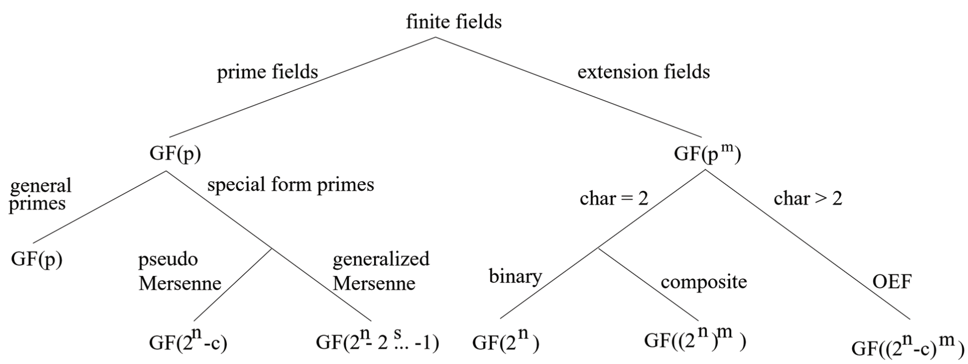

Fig. 7. Finite fields proposed for use in public-key schemes [Paar 1999]. Of recent interest to cryptographers are $\mathbb{F}_p$ and $\mathbb{F}_{2^p}$, where p is prime, as neither appears vulnerable to subexponential attack [Gaudry *et al.* 2000].

of ECDLP to DLP over hyperelliptic curves [Gaudry *et al.* 2000]. But $\mathbb{F}_{2^p}$, a binary extension field, remains popular among implementations of ECC, especially those in hardware, as it allows for particularly space- and time-efficient algorithms. In light of its applications in coding, the field has also received more attention in the literature than those of other characteristics [Paar 1999].

It was with this history in mind that we proceeded with our implementation of ECC over $\mathbb{F}_{2^p}$ toward an end of smaller public keys for the MICA2.

4.1 Elliptic Curves over $\mathbb{F}_{2^p}$

It turns out that, over $\mathbb{F}_{2^p}$, Equation 1 simplifies to

$$y^2 + xy \equiv x^3 + ax^2 + b, \tag{2}$$

where $a, b \in \mathbb{F}_{2^p}$, upon substitution of $a_1^2 x + \frac{a_3}{a_1}$ for x and $a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3}$ for y, if we consider only nonsupersingular curves, for which $a_1 \neq 0$. It is the set of solutions to Equation 2 and, more generally, Equation 1 (*i.e.*, the points on E), that actually provides the foundation for smaller public keys on the MICA2. All that remains is specification of some algebraic structure over that set. An Abelian group suffices but requires provision of some binary operator offering closure, associativity, identity, inversion, and commutativity. As suggested by ECDLP's definition, that operator is to be addition.

The addition of two points on a curve over $\mathbb{F}_{2^p}$ is defined as

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3),$$

such that

$$(x_3, y_3) = (\lambda^2 + \lambda + x_1 + x_2 + a,\ \lambda(x_1 + x_3) + x_3 + y_1),$$

where

$$\lambda = (y_1 + y_2)(x_1 + x_2)^{-1}.$$

However, so that the group is Abelian, it is necessary to define a "point at infinity," $\mathcal{O}$, whereby

$$\begin{aligned} \mathcal{O} + \mathcal{O} &= \mathcal{O}, \\ (x, y) + \mathcal{O} &= (x, y), \text{ and} \\ (x, y) + (x, -y) &= \mathcal{O}. \end{aligned}$$

Doubling of some point, meanwhile, is defined as

$$(x_1, y_1) + (x_1, y_1) = (x_3, y_3),$$

such that

$$(x_3, y_3) = (\lambda^2 + \lambda + a,\ x_1^2 + (\lambda + 1)x_3),$$

where

$$\lambda = x_1 + y_1 x_1^{-1},$$

provided $x_1 \neq 0$.

With these primitives is point multiplication also possible [Gordon 1998]. With an algebraic structure on the points of elliptic curves over $\mathbb{F}_{2^p}$ thus defined, implementation of a cryptosystem is possible.

4.2 ECC over $\mathbb{F}_{2^p}$

Implementation of ECC over $\mathbb{F}_{2^p}$ first requires a choice of basis for points' representation, insofar as each $a \in \mathbb{F}_{2^p}$ can be written as

$$a = \sum_{i=0}^{m-1} a_i \alpha_i,$$

where $a_i \in \{0, 1\}$. Thus defined, a can be represented as a binary vector, $\{a_0, a_1, \ldots, a_{p-1}\}$, where $\{\alpha_0, \alpha_1, \ldots, \alpha_{p-1}\}$ is its basis over $\mathbb{F}_2$. Most common for bases over $\mathbb{F}_2$ are polynomial bases and normal bases, though dual, triangular, and other bases exist.

When represented with a polynomial basis, each $a \in \mathbb{F}_{2^p}$ corresponds to a binary polynomial of degree less than p, whereby

$$a = a_{p-1}x^{p-1} + a_{p-2}x^{p-2} + \cdots + a_0 x^0,$$

where, again, $a_i \in \{0,1\}$. Accordingly, each $a \in \mathbb{F}_{2^p}$ can be represented in the MICA2's SRAM as a bit string, $a_{p-1}a_{p-2}\cdots a_0$. All operations on these elements are performed modulo an irreducible reduction polynomial, f, of degree p over $\mathbb{F}_2$, such that $f(x) = x^p + \sum_{i=0}^{p-1} f_i x_i$, where $f_i \in \{0,1\}$ for $i \in \{0,1,\ldots,p-1\}$. Typically, if an irreducible trinomial, $x^p + x^k + 1$, exists over $\mathbb{F}_{2^p}$, then $f(x)$ is chosen to be that with smallest k; if no such trinomial exists, then $f(x)$ is chosen to be a pentanomial, $x^p + x^{k_3} + x^{k_2} + x^{k_1} + 1$, such that k_1 is minimal, k_2 is minimal given k_1, and k_3 is minimal given k_1 and k_2 [López and Dahab 2000a].

In a polynomial basis, addition of two elements, a and b is defined as $a + b = c$, where $c_i \equiv a_i + b_i \pmod 2$ (*i.e.*, a sequence of XORs). Multiplication of a and b, meanwhile, is defined as $a \cdot b = c$, where $c(x) \equiv (\sum_{i=0}^{p-1} a_i x^i)(\sum_{i=0}^{p-1} b_i x^i) \pmod{f(x)}$.

We selected a polynomial basis for our implementations of point multiplication on the MICA2, as it tends to allow for more efficient implementations in software [Barwood 1997].

4.3 First Implementation

Our first implementation of ECC on the MICA2 (EccM 1.0), a TinyOS module based on code by Michael Rosing [Rosing 1999], whose *Implementing Elliptic Curve Cryptography* is a popular starting point for any implementation of ECC, ultimately reinforced prevailing wisdom: it was a failure.

EccM 1.0 first selected a random curve in the form of Equation 2, such that $a = 0$ and $b \in \mathbb{F}_{2^p}$. It next selected a random point, $G \in \mathbb{F}_{2^p} \times \mathbb{F}_{2^p}$, from that curve as well as a random $k \in \mathbb{F}_{2^p}$, the node's private key. Finally, it computed $k \cdot G$, the node's public key.

As in Rosing's code, this implementation employed a number of optimizations. Addition of points was implemented in accordance with Schroeppel *et al.* [Schroeppel *et al.* 1995]; multiplication of points followed Koblitz [Koblitz 1992]; conversion of integers to non-adjacent form was accomplished as in Solinas [Solinas 1997]. Generation of pseudorandom numbers, meanwhile, was achieved with Marsaglia [Marsaglia 1994].

On first glance, the results (Fig. 8) were encouraging, with generation of 33-bit keys requiring just 1.776 sec. (The module itself performed these measurements.) Unfortunately, for larger keys (*e.g.*, 63-bit), the module failed to produce results, instead causing the mote to reset as a result of stack overflow. Although none of the module's functions were recursive, several utilized a good deal of memory for multi-word arithmetic. Fig. 9 offers the results of an analysis of EccM 1.0's usage of SRAM, determined with stacktool [Regehr 2004].

4.4 Second Implementation

Since optimizations of EccM 1.0 failed to render generation of even 63-bit keys possible, an overhaul of this popular implementation proved necessary for realization of 163-bit keys. Inspired by the design of Dragongate Technologies Limited's Java-based jBorZoi 0.9 [Dragongate Technologies Limited 2003], EccM 2.0 similarly implements ECC but with far greater success. EccM 2.0 selects for a node, Alice, a private key, k_A, using a polynomial basis over $\mathbb{F}_{2^p}$, thereafter computing with a Koblitz curve and base point, G, Alice's public key, T_A. Alice's public key is then broadcasted (in two, 22-byte payloads) to any node, Bob, with whom secure

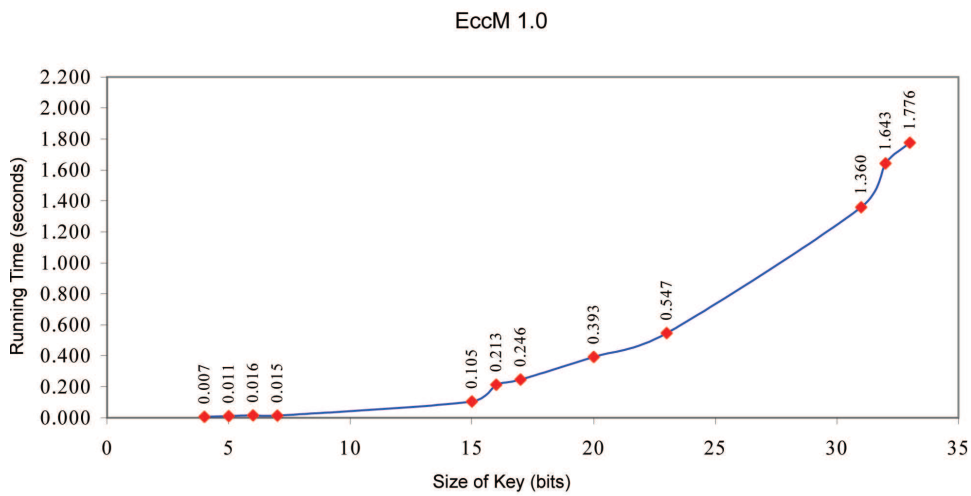

Fig. 8. Running time for EccM 1.0, a TinyOS module that selected for a node at random, using a polynomial basis over $\mathbb{F}_{2^p}$, a curve, a point, and a private key, thereafter computing the node's public key. Points are labelled with running times. For larger keys (*e.g.*, 63-bit), the module failed to produce results.

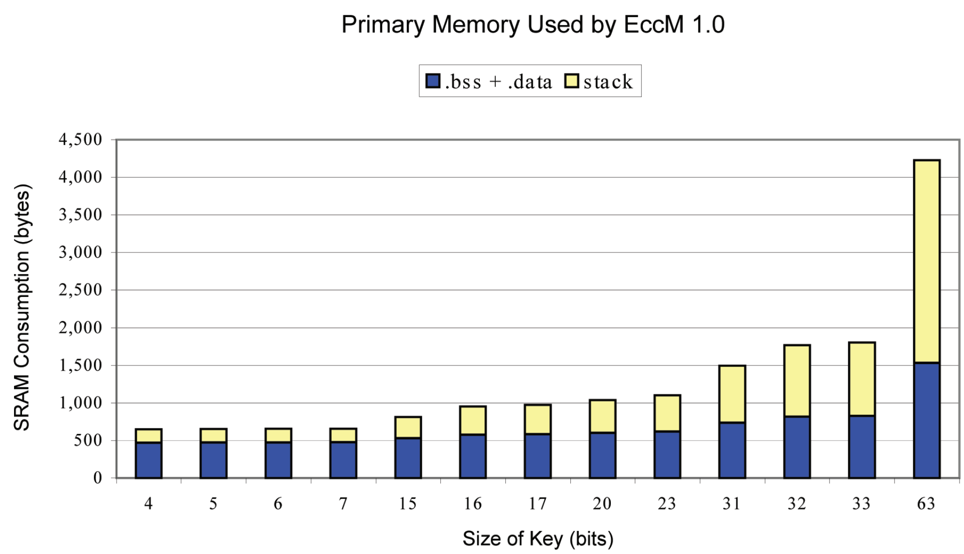

Fig. 9. Primary memory used by EccM 1.0, a TinyOS module that, using a polynomial basis over $\mathbb{F}_{2^p}$, selected for a node at random a curve, a point, and a private key, thereafter computing the node's public key. Although the sizes of the `.bss` and `.data` segments are fixed during execution, stack is defined here as the maximum of the application's stack size during execution. Keys of 63 bits or more exhaust the MICA2's 4,096 KB of SRAM.

communication is desired. Provided Alice receives Bob's public key, T_B, from Bob in this same manner, each can compute a shared secret, $k_A \cdot k_B \cdot G$, where k_B is Bob's private key. If so desired, this secret could be massaged into compliance with a standard like the Elliptic Curve Key Agreement Scheme, Diffie-Hellman 1 (ECKAS-DH1) [IEEE Computer Society 2000].

In EccM 2.0, multiplication of points is achieved with Algorithm IV.1 in Blake *et al.* [Blake *et al*. 1999], while addition of points is achieved with Algorithm 7 in López and Dahab [López and Dahab 2000a]. Multiplication of elements in $\mathbb{F}_{2^p}$, meanwhile, is implemented as Algorithm 4 in López and Dahab [López and Dahab 2000b], while inversion is implemented as Algorithm 8 in Hankerson *et al.* [Hankerson *et al*. 2001].

Beyond rendering 163-bit public keys feasible, EccM 2.0 also redresses another shortcoming in EccM 1.0. Inasmuch as EccM 1.0 selects curves at random, it risks (albeit with exponentially small probability) selection of supersingular curves that are vulnerable to sub-exponential attack via MOV reduction [Menezes *et al*. 1991] with index-calculus methods [Silverman and Suzuki 1998]. EccM 2.0 thus obeys NIST's recommendation for ECC over $\mathbb{F}_{2^p}$ [National Institute of Standards and Technology 1999], selecting, for the results herein,

$$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$$

for the reduction polynomial,

$$y^2 + xy \equiv x^3 + x^2 + 1$$

for the curve, E, the order of (*i.e.*, number of points on) which is 0x40000000000000000000020108a2e0cc0d99f8a5ef, and, for the point $G = (G_x, G_y)$,

$$G_x = \text{0x2fe13c0537bbc11acaa07d793de4e6d5e5c94eee8}$$

and

$$G_y = \text{0x289070fb05d38ff58321f2e800536d538ccdaa3d9}.$$

Ultimately, not only does EccM 2.0 employ much less memory than does EccM 1.0 (Table VIII), per stacktool [Regehr 2004], its running time bests that for Diffie-Hellman based on DLP, using keys an order of magnitude smaller in size but no less secure. (The module itself measures the times required to generate keys and to generate shared secrets.) The time required to generate a private and public key pair with this module, averaged over 100 trials, is just 34.161 sec, with a standard deviation of 0.921 sec. The time required to generate a shared secret, given one's private key and another's public key, averaged over 100 trials, is 34.173 sec, with a standard deviation of 0.934 sec. In short, distribution of some shared secret using ECC over $\mathbb{F}_{2^p}$ requires no more than a minute or so of computation per node in total. Table IX details the module's energy consumption, measured as before for Diffie-Hellman over DLP. Although such performance might prove unacceptable for some applications of PKI, it appears quite reasonable for infrequent distribution of TinySec keys. Moreover, as more recent work confirms (Section 6), it is an upper bound on the time required.

Since the release of its source code, EccM 2.0 has been incorporated into or been a point of comparison for a variety of projects [Arazi and Qi 2006; Blaß and

	EccM 1.0 (32-bit key)	**EccM 2.0 (163-bit key)**
`.bss`	826 B	1,055 B
`.data`	6 B	4 B
`.text`	17,544 B	34,342 B
stack	976 B	81 B

Table VIII. Memory usage of EccM 1.0 versus EccM 2.0. With EccM 2.0, we obtain significantly more bits of security using a reasonable footprint in memory. The `.bss` and `.data` segments consume SRAM while the `.text` segment consumes ROM. Stack is defined here as the maximum of the application's stack size during execution. Much of the increase of ROM's consumption is the result of EccM 2.0's additional functionality.

	Private-Key Generation	**Public-Key Generation**
Total Time	0.229 sec	34.161 sec
Total CPU Utilization	1.690×10^6 cycles	2.512×10^8 cycles
Total Energy	0.00549 Joules	0.816 Joules

Table IX. Energy consumption of EccM 2.0, a TinyOS module that allows two nodes to generate public and private keys (and, thereafter, to use the same to exchange a shared secret), during generation of a node's public and private keys.

Zitterbart 2005; Benenson *et al*. 2005; Deng *et al*. 2006; Rochester Institute of Technology 2005; Seo *et al*. 2006; Wang and Li 2006]. A link to EccM 2.0's source code is offered toward this paper's end.

5. DISCUSSION

EccM 2.0's average running time of, roughly, 34 seconds for point multiplication was the result of several iterations of optimization. In fact, this module initially clocked 7.782 minutes for this computation, well beyond any reasonable bound. To be sure, we spent some cycles foolishly (*e.g.*, unnecessarily recomputing the terminal condition for some loop). But other waste was less obvious. Apparent only to us (and not to nesC's compiler), certain loops were simply better off iterating from high to low than from low to high, given the expected lengths of various multi-precision intermediates. Other loops proved better off once manually unrolled.

Rather than handle multi-precision bit shifts with a generalized implementation, we were able to shave seconds off the running time by special-casing the most common of shifts (namely left shifts by one bit and by two bits), albeit at a cost of a larger `.bss` segment.

Consider that, with inlining disabled, even the second version of this module induced hundreds of thousands of function calls, largely the result of the module's requirement of multi-precision arithmetic. Even the slightest of improvements in some function's performance, then, can effect significant gains overall.

Other optimizations were grounded in published, theoretical results. Substitution of Algorithm 2 in Hankerson *et al.* [Hankerson *et al*. 2001] with Algorithm 4 in López and Dahab [López and Dahab 2000b] offered several seconds of improvement, as did implementation of Algorithm 7 in López and Dahab [López and Dahab 2000a]. But the art of source-level, hand optimizations, so infrequently deployed for modern systems, proved remarkably helpful, daresay necessary, for an environment so constrained as the MICA2.

6. RELATED WORK

Studied by mathematicians for more than a century, elliptic curves claim significant coverage in the literature. ECC, meanwhile, has received much attention since its discovery in 1985.

Since completion of our earliest work [Malan 2004], the viability of PKI for sensor networks has received significantly more attention. Gura *et al.* offer significant improvement over our earlier results using $\mathbb{F}_p$ instead of $\mathbb{F}_{2^p}$ [Gura *et al*. 2004; Wander *et al*. 2005]. Ning and Liu now offer TinyECC 0.1 [Ning and Liu 2005], which Wang compares to EccM 2.0 [Wang and Li 2006], while Gupta *et al.* offer Sizzle (ECC-based SSL) [Gupta *et al*. 2005], both over $\mathbb{F}_p$ [Ning and Liu 2005]. Du *et al.* investigate alternatives to expensive public-key operations [Du *et al*. 2005]. Gaubatz *et al.* propose a hardware-assisted approach to PKI [Gaubatz *et al*. 2004]. Benenson *et al.*, meanwhile, implement digital signatures atop EccM 2.0 [Benenson *et al*. 2005]. Watro *et al.*, on the other hand, explore RSA as a PKI's foundation [Watro *et al*. 2004].

Though less recent, of particular relevance to our work is Woodbury's recommendation of an optimal extension field, $\mathbb{F}_{(2^8-17)^{17}}$, for low-end, 8-bit processors [Woodbury 2001]. Ernst *et al.* propose supplementary hardware for AVR implementing operations over binary fields [Ernst *et al*. 2002]. Handschuh and Paillier propose cryptographic coprocessors for smart cards [Handschuh and Paillier 2000], whereas Woodbury *et al.* describe ECC for smart cards without coprocessors [Woodbury *et al*. 2000]. Albeit for a different target, Hasegawa *et al.* provide a "small and fast" implementation of ECC in software over $\mathbb{F}_p$ for a 16-bit microcomputer [Hasegawa *et al*. 1999]. Messerges *et al.* call for ECC with 163-bit keys for mobile, *ad hoc* networks [Messerges *et al*. 2003]. Guajardo *et al.* describe an implementation of ECC for the 16-bit TI MSP430x33x family of microcontrollers [Guajardo *et al*. 2001]. Weimerskirch *et al.*, meanwhile, offer an implementation of ECC for Palm OS [Weimerskirch *et al*. 2001], and Brown *et al.* offer the same for Research In Motion's RIM pager [Brown *et al*. 2000].

ZigBee, on the other hand, shares this work's aim of wireless security for sensor networks albeit not with ECC but with AES-128 [ZigBee Alliance 2004], a shared-key protocol. Meanwhile, recommendations for ECC's parameters abound, among academics [Lenstra and Verheul 1999], among corporations [Certicom Corporation 2004], and within government [IEEE Computer Society 2000; National Institute of Standards and Technology 1999].

A number of implementations of ECC in software are freely available in languages other than nesC. Rosing [Rosing 1999] offers his C-based implementation of ECC over $\mathbb{F}_{2^p}$ with both polynomial and normal bases. ECC-LIB [Zaroliagis 2004] and pegwit [Barwood 2006] offer their own C-based implementations over $\mathbb{F}_{2^p}$ with polynomial bases. MIRACL [Shamus Software Ltd 2004] provides the same, with an additional option for curves over $\mathbb{F}_p$. LibTomCrypt [Denis 2004], also in C, focuses on $\mathbb{F}_p$. Dragongate Technologies Limited, meanwhile, offers borZoi and jBorZoi [Dragongate Technologies Limited 2003], implementations of ECC over $\mathbb{F}_{2^p}$ with polynomial bases in C++ and Java, respectively. Another implementation in C++, also using a polynomial basis over $\mathbb{F}_{2^p}$, is available through libecc [Wood 2004].

7. FUTURE WORK

Opportunities for future work certainly remain. Reduction of EccM 2.0's ROM requirements is certainly of interest, as the module currently consumes a non-trivial amount (34 KB) of the MICA2's 128-KB ROM. Optimizations of the module's nesC source code might allow us to reclaim some of those bytes; re-implementation of one or more functions in AVR assembly might allow us to reclaim even more. Of course, as expectations of secure communications rise, cryptographic primitives like those in TinySec and EccM 2.0 could simply be integrated into hardware (much like Texas Instruments has done with AES in its CC2420 transceiver [Texas Instruments 2007]), thereby reserving motes' own resources for actual applications.

Further reduction of EccM 2.0's running time, through source- or assembly-level enhancements, is also of interest, wherever the module happens to be housed, particularly in light of others' recent results [Gura *et al*. 2004; Gupta *et al*. 2005; Ning and Liu 2005; Wander *et al*. 2005]. Use of wNAF (width non-adjacent form) or wMOF (width mutual opposite form) might also help to reduce our numbers of scalar multiplications [Okeya *et al*. 2004a].[5] Worthy of consideration for future versions of this module is a normal basis, an advantage of which would be its implementation using only ANDs, XORs, and cyclic shifts, beneficiaries of which are multiplication and squaring. (For this reason, normal bases tend to be popular in implementations of ECC in hardware.) Of value as well might be a hybrid of polynomial and normal bases, as such is thought to leverage advantages of each simultaneously [Rosing 1999].

Of course, work by Gura *et al.* [Gura *et al*. 2004] and Gupta *et al.* [Gupta *et al*. 2005] suggests that the module might offer even better performance if re-implemented over $\mathbb{F}_p$, especially as expensive inversions could be avoided through use of projective (as opposed to affine) coordinates [Grosschädl 2003]. Although relatively efficient algorithms exist for modular reduction (*e.g.*, those of Montogomery [Montgomery 1985] or Barrett [Barrett 1987]), selection of a generalized Mersene number for p would also allow modular reduction to be executed as a more efficient sequence of three additions (mod p) [Solinas 1999]. Also of potential benefit are mixed coordinates [Cohen *et al*. 1998], non-adjacent form [Miller 1986b], mutual opposite form [Okeya *et al*. 2004b], fractional windows [Möller 2004], and signed binary representations [Kong and Li 2005; Joye and Yen 2000].

Performance aside, EccM 2.0's reliance on TinyOS's RandomLFSR module is troubling cryptographically, as this pseudo-random number generator (PRNG) relies solely upon a mote's unique ID for seeding, rather than upon any physical source of randomness. Implementation of a superior PRNG is necessary for our module's security. Truly random bits might be captured from such sources as local sensor readings, interrupt and packet-arrival times, and other physical sources.

[5]Personal correspondence with Seog Chung of the Gwangju Institute of Science and Technology.

It also remains to define the protocol according to which a module like EccM 2.0 would operate to rekey nodes. Prerequisite, for instance, will be some form of authentication, lest adversaries be able to trigger rekeyings, thereby sapping motes' energy and otherwise interfering with communication.

8. CONCLUSION

Despite claims to the contrary, public-key infrastructure is viable on the MICA2, certainly for infrequent distribution of shared secrets. Although our implementation of ECC in 4 KB of primary memory on this 8-bit, 7.3828-MHz device offers room for further optimization, its successors corroborate and demonstrate ECC's viability as a foundation for PKI for sensor networks.

The need for PKI's success on the MICA2 seems clear. TinySec's shared secrets do allow for efficient, secure communications among nodes. But such devices as those in sensor networks, for which physical security is unlikely, require some mechanism for secret keys' distribution.

In that it offers equivalent security at lower cost to memory and bandwidth than does Diffie-Hellman based on DLP, a public-key infrastructure for key distribution based on elliptic curves is an apt and increasingly viable choice for sensor networks.

SOURCE CODE

Source code for EccM 2.0, BenchmarksM (plus its `MessageListener`), and our instrumented TinySecM is available for download from `http://www.eecs.harvard.edu/~malan/`.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 310877. Many thanks to Breanne Duncan, Mark Hempstead, Glenn Holloway, Michael Mitzenmacher, and Victor Shnayder of Harvard University and to David van Dyk of the University of California at Irvine for their assistance with this work.

REFERENCES

ADLEMAN, L. M. 1979. A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography. In *Proc. 20th IEEE Found. Comp. Sci. Symp.* 55–60.

ARAZI, O. AND QI, H. 2006. Load-Balanced Key Establishment Methodologies in Wireless Sensor Networks. In *International Journal of Security and Networks*. Vol. 1.

BARRETT, P. 1987. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology – CRYPTO '86*, A. M. Odlyzko, Ed. Vol. 263.

BARWOOD, G. 1997. Elliptic Curve Cryptography FAQ v1.12 22nd. `http://www.cryptoman.com/elliptic.htm`.

BARWOOD, G. 2006. Pegwit (v8). `http://www.george-barwood.pwp.blueyonder.co.uk/hp/v8/pegwit.htm`.

BBN TECHNOLOGIES. 2003. Diffie-Hellman 1.

BENENSON, Z., GEDICKE, N., AND RAIVIO, O. 2005. Realizing Robust User Authentication in Sensor Networks. In *REALWSN'05: Workshop on Real-World Wireless Sensor Networks*. Stockholm, Sweden.

BIHAM, E., BIRYUKOV, A., AND SHAMIR, A. 1999. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. *Lecture Notes in Computer Science 1592*, 12–23.

BLAKE, I., SEROUSSI, G., AND SMART, N. 1999. Elliptic Curves in Cryptography. *LMS Lecture Note Series 265*.

BLASS E.-O. AND ZITTERBART, M. 2005. Towards Acceptable Public-Key Encryption in Sensor Networks. In *IWUC 2005: Proceedings of the First International Workshop on Ubiquitous Computing*.

BROWN, M., CHEUNG, D., HANKERSON, D., HERNANDEZ, J. L., KIRKUP, M., AND MENEZES, A. 2000. PGP in Constrained Wireless Devices. In *Proceedings of the 9th USENIX Security Symposium*. USENIX Association.

CERPA, A., ELSON, J., ESTRIN, D., GIROD, L., HAMILTON, M., AND ZHAO, J. 2001. Habitat monitoring: Application Driver for Wireless Communications Technology.

CERTICOM CORPORATION. 2000. Remarks On The Security Of The Elliptic Curve Cryptosystem. `http://www.comms.engg.susx.ac.uk/fft/crypto/EccWhite3.pdf`.

CERTICOM CORPORATION. 2004. Standards for Efficient Cryptography Group. `http://www.secg.org/`.

COHEN, H., MIYAJI, A., AND ONO, T. 1998. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In *ASIACRYPT '98: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*. Springer-Verlag, London, UK, 51–65.

CROSSBOW TECHNOLOGY, INC. 2004. MICA2: Wireless Measurement System. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-04_A_MICA2.pdf`.

DENG, J., HAN, R., AND MISHRA, S. 2006. Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*. ACM Press, New York, NY, USA, 292–300.

DENIS, T. S. 2004. LibTomCrypt. `http://libtomcrypt.org/`.

DIFFIE, W. AND HELLMAN, M. E. 1976. New Directions in Cryptography. *IEEE Transactions on Information Theory IT-22,* 6, 644–654.

DIFFIE, W., VAN OORSCHOT, P. C., AND WIENER, M. J. 1992. Authentication and Authenticated Key Exchanges. *Designs, Codes, and Cryptography 2,* 2, 107–125.

DRAGONGATE TECHNOLOGIES LIMITED. 2003. jBorZoi 0.9. `http://dragongate-technologies.com/products.html`.

DU, W., WANG, R., AND NING, P. 2005. An Efficient Scheme for Authenticating Public Keys in Sensor Networks. In *MobiHoc '05: Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM Press, New York, NY, USA, 58–67.

ERNST, M., JUNG, M., MADLENER, F., HUSS, S., AND BLÜMEL, R. 2002. A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2^n). In *CHES*. 381–399.

EVERYREADY BATTERY COMPANY. 2004. Engineering Datasheet: Energizer No. X91. `http://data.energizer.com/datasheets/library/primary/alkaline/energizer/consumer_oem/e91.pdf`.

FREY, G. AND GANGL, H. 1998. How to Disguise an Elliptic Curve (Weil Descent). ECC '98.

GAUBATZ, G., KAPS, J.-P., AND SUNAR, B. 2004. Public Key Cryptography in Sensor Networks—Revisited. In *ESAS '04: Proceedings of the 1st European Workshop on Security in Ad-hoc and Sensor Networks*. Vol. LNCS3313. Springer, 2–18.

GAUDRY, P., HESS, F., AND SMART, N. P. 2000. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. Tech. Rep. CSTR-00-016, Department of Computer Science, University of Bristol. October.

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.

GORDON, D. M. 1993. Discrete Logarithms in GF(P) Using the Number Field Sieve. *SIAM J. Discret. Math. 6,* 1, 124–138.

GORDON, D. M. 1998. A Survey of Fast Exponentiation Methods. *J. Algorithms 27,* 1, 129–146.

GROSSSCHÄDL, J. 2003. Implementation Options for Elliptic Curve Cryptography.

Guajardo, J., Blümel, R., Krieger, U., and Paar, C. 2001. Efficient Implementation of Elliptic Curve Cryptosystems on the TI MSP430x33x Family of Microcontrollers. In *PKC 2001*, K. Kim, Ed. Korea, 365–382.

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. In *PerCom*. 247–256.

Gura, N., Patel, A., Wander, A., Eberle, H., and Shantz, S. C. 2004. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. 6th International Workshop on Cryptographic Hardware and Embedded Systems, Boston, Massachusetts.

Handschuh, H. and Paillier, P. 2000. Smart Card Crypto-Coprocessors for Public-Key Cryptography. In *Lecture Notes in Computer Science*, J.-J. Quisquater and B. Schneier, Eds. Smart Card Research and Applications. Springer-Verlag, 386–394.

Hankerson, D., Hernandez, J. L., and Menezes, A. 2001. Software Implementation of Elliptic Curve Cryptography over Binary Fields. *Lecture Notes in Computer Science 1965*.

Hasegawa, T., Nakajima, J., and Matsui, M. 1999. A Small and Fast Software Implementation of Elliptic Curve Cryptosystems over GF(p) on a 16-Bit Microcomputer. *IEICE Trans. Fundamentals E82-A,* 1 (January), 98–106.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. E., and Pister, K. S. J. 2000. System Architecture Directions for Networked Sensors. In *Architectural Support for Programming Languages and Operating Systems*. 93–104.

IEEE Computer Society. 2000. *IEEE Standard Specifications for Public-Key Cryptography.*

Joye, M. and Yen, S.-M. 2000. Optimal Left-to-Right Binary Signed-Digit Recoding. *IEEE Transactions on Computers 49,* 7, 740–748.

Karlof, C., Sastry, N., and Wagner, D. 2004a. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. Second ACM Conference on Embedded Networked Sensor Systems, Baltimore, Maryland.

Karlof, C., Sastry, N., and Wagner, D. 2004b. TinySec: Link Layer Security for Tiny Devices. `http://www.cs.berkeley.edu/~nks/tinysec/`.

Koblitz, N. 1987. Elliptic Curve Cryptosystems. *Mathematics of Computation 48*, 203–209.

Koblitz, N. 1992. CM-Curves with Good Cryptographic Properties. In *Advances in Cryptology – CRYPTO '91*. 279–287.

Kong, F. and Li, D. 2005. A Note on Signed Binary Window Algorithm for Elliptic Curve Cryptosystems. In *CANS*. 223–235.

Kottapalli, V. A., Kiremidjian, A. S., Lynch, J. P., Carryer, E., Kenny, T. W., Law, K. H., and Lei, Y. 2003. Two-Tiered Wireless Sensor Network Architecture for Structural Health Monitoring. SPIE's 10th Annual International Symposium on Smart Structures and Materials.

LaMacchia, B. A. and Odlyzko, A. M. 1991. Computation of Discrete Logarithms in Prime Fields. *Lecture Notes in Computer Science 537*, 616–618.

Lenstra, A. K. and Verheul, E. R. 1999. Selecting Cryptographic Key Sizes.

López, J. and Dahab, R. 2000a. An Overview of Elliptic Curve Cryptography. Tech. rep., Institute of Computing, Sate University of Campinas, São Paulo, Brazil. May.

López, J. and Dahab, R. 2000b. High-Speed Software Multiplication in $\mathbb{F}_{2^m}$. Tech. rep., Institute of Computing, Sate University of Campinas, São Paulo, Brazil. May.

Malan, D. 2004. Crypto for Tiny Objects. Tech. Rep. TR-04-04, Harvard University, Cambridge, Massachusetts, USA. January.

Malan, D., Fulford-Jones, T., Welsh, M., and Moulton, S. 2004. CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In *International Workshop on Wearable and Implantable Body Sensor Networks*. London, United Kingdom.

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 *Proceedings of the First IEEE International Conference on Sensor and Ad Hoc Communications and Networks*. Santa Clara, California.

Marsaglia, G. 1994. The Mother of All Random Generators. ftp://ftp.taygeta.com/pub/c/mother.c.

Menezes, A., Vanstone, S., and Okamoto, T. 1991. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*. ACM Press, 80–89.

Messerges, T. S., Cukier, J., Kevenaar, T. A. M., Puhl, L., Struik, R., and Callaway, E. 2003. A Security Design for a General Purpose, Self-Organizing, Multihop Ad Hoc Wireless Network. ACM Workshop on Security of Ad Hoc and Sensor Networks, George Mason University, Fairfax, Virginia.

Miller, V. 1986a. Uses of Elliptic Curves in Cryptography. In *Lecture Notes in Computer Science 218: Advances in Crytology - CRYPTO '85*. Springer-Verlag, Berlin, 417–426.

Miller, V. S. 1986b. Use of Elliptic Curves in Cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*. Springer-Verlag New York, Inc., New York, NY, USA, 417–426.

Möller, B. 2004. Fractional Windows Revisited: Improved Signed-Digit Representations for Efficient Exponentiation. In *ICISC*. 137–153.

Montgomery, P. 1985. Modular Multiplication without Trial Division. *Mathematics of Computation 44,* 170, 519–521.

National Institute of Standards and Technology. 1988. *SKIPJACK and KEA Algorithm Specifications*. Computer Security Division.

National Institute of Standards and Technology. 1994. Federal Information Processing Standards Publication 185: Escrowed Encryption Standard (EES).

National Institute of Standards and Technology. 1999. Recommended Elliptic Curves For Federal Government Use. http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf.

National Institute of Standards and Technology. 2003. Special Publication 800-57: Recommendation for Key Management.

NEST Challenge Architecture. 2002.

Ning, P. and Liu, A. 2005. TinyECC: Elliptic Curve Cryptography for Sensor Networks. http://discovery.csc.ncsu.edu/software/TinyECC/.

Okeya, K., Schmidt-Samoa, K., Spahn, C., and Takagi, T. 2004a. *Lecture Notes in Computer Science*. Vol. 3152. Springer Berlin / Heidelberg, Chapter Signed Binary Representations Revisited, 123.

Okeya, K., Schmidt-Samoa, K., Spahn, C., and Takagi, T. 2004b. Signed Binary Representations Revisited. Cryptology ePrint Archive, Report 2004/195. http://eprint.iacr.org/.

Paar, C. 1999. Implementation Options for Finite Field Arithmetic for Elliptic Curve Cryptosystems. Invited presentation at the 3rd Workshop on Elliptic Curve Cryptography (ECC '99).

Perlman, R. 2003. Computer Science 243, Harvard University.

Perrig, A., Stankovic, J., and Wagner, D. 2004. Security in Wireless Sensor Networks. *Communications of the ACM 47,* 6 (June), 53–57.

Perrig, A., Szewczyk, R., Wen, V., Culler, D. E., and Tygar, J. D. 2001. SPINS: Security Protocols for Sensor Networks. In *Mobile Computing and Networking*. 189–199.

Rabin, M. 1979. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Tech. Rep. MIT/LCS/TR-212, MIT.

Regehr, J. 2004. John Regehr's Stack Bounding Page. http://www.cs.utah.edu/~regehr/stacktool/.

Rochester Institute of Technology. 2005. CISCO University Research Program Project. http://www.ce.rit.edu/~fxheec/cisco_urp/docs/Main_ECC_Doc.htm.

Rosing, M. 1999. *Implementing Elliptic Curve Cryptography*. Manning Publications Co.

Schroeppel, R., Orman, H., O'Malley, S., and Spatscheck, O. 1995. Fast Key Exchange with Elliptic Curve Systems. *Lecture Notes in Computer Science 963*.

SEO, S. C., KIM, H. C., AND RAMAKRISHNA, R. S. 2006. A New Security Protocol Based on Elliptic Curve Cryptosystems for Security Wireless Sensor Networks. In *SECUBIQ 2006: Proceedings of the Second International Workshop on Security in Ubiquitous Computing Systems*.

SHAMUS SOFTWARE LTD. 2004. Multiprecision Integer and Rational Arithmetic C/C++ Library. `http://indigo.ie/~mscott/#Elliptic`.

SHNAYDER, V., HEMPSTEAD, M., RONG CHEN, B., ALLEN, G. W., AND WELSH, M. 2004. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *SenSys '04: Proceedings of the Second International Conference on Embedded Networked Sensor Systems*. ACM Press, New York, NY, USA, 188–200.

SILVERMAN, J. AND SUZUKI, J. 1998. Elliptic Curve Discrete Logarithms and the Index Calculus. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag.

SOLINAS, J. 1999. Generalized Mersenne Numbers. Tech. Rep. CORR-39, University of Waterloo.

SOLINAS, J. A. 1997. An Improved Algorithm for Arithmetic on a Family of Elliptic Curves. In *Advances in Cryptology – CRYPTO '97*. 357–371.

TEXAS INSTRUMENTS. 2007. 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B).

WANDER, A. S., GURA, N., EBERLE, H., GUPTA, V., AND SHANTZ, S. C. 2005. Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*. IEEE Computer Society, Washington, DC, USA, 324–328.

WANG, H. AND LI, Q. 2006. *International Journal of Security and Networks*. Vol. 1. Chapter Elliptic Curve Cryptography Based Access Control in Sensor Networks.

WATRO, R. 2003. Lightweight Security for Wireless Networks of Embedded Systems. `http://www.is.bbn.com/projects/lws-nest/bbn_nest_apr_03.ppt`.

WATRO, R., KONG, D., FEN CUTI, S., GARDINER, C., LYNN, C., AND KRUUS, P. 2004. TinyPK: Securing Sensor Networks with Public Key Technology. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*. ACM Press, New York, NY, USA, 59–64.

WEIMERSKIRCH, A., PAAR, C., AND SHANTZ, S. C. 2001. Elliptic Curve Cryptography on a Palm OS Device. The 6th Australasian Conference on Information Security and Privacy, Sydney, Australia.

WOOD, C. 2004. libecc. `http://libecc.sourceforge.net/`.

WOODBURY, A. D. 2001. Efficient Algorithms for Elliptic Curve Cryptosystems on Embedded Systems. `http://www.wpi.edu/Pubs/ETD/Available/etd-1001101-195321/unrestricted/woodbury.pdf`.

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 (CARDIS 2000) Conference, Bristol, UK.

ZAROLIAGIS, C. 2004. ECC-LIB: A Library for Elliptic Curve Cryptography. `http://www.ceid.upatras.gr/faculty/zaro/software/ecc-lib/`.

ZIGBEE ALLIANCE. 2004. `http://www.zigbee.org/`.