Crypto for Tiny Objects

David Malan

TR-04-04

2004

Computer Science Group
Harvard University
Cambridge, Massachusetts

Crypto for Tiny Objects

David Malan
malan@eecs.harvard.edu

Abstract

This work presents the first known implementation of elliptic curve cryptography for sensor networks, motivated by those networks' need for an efficient, secure mechanism for shared cryptographic keys' distribution and redistribution among nodes. Through instrumentation of UC Berkeley's TinyOS, this work demonstrates that secret-key cryptography is already viable on the MICA2 mote. Through analyses of another's implementation of modular exponentiation and of its own implementation of elliptic curves, this work concludes that public-key infrastructure may also be tractable in 4 kilobytes of primary memory on this 8-bit, 7.3828-MHz device.

1 Introduction

Wireless sensor networks have been proposed for such applications as habitat monitoring [15], structural health monitoring [38], emergency medical care [6], and vehicular tracking [8], all of which demand some combination of authentication, integrity, privacy, and security. Unfortunately, the state of the art offers 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 of implementations of these needs. Consider the MICA2 mote [3], designed by researchers at the University of California at Berkeley and fabricated by Crossbow Technology, Inc.: supported by Berkeley's TinyOS operating system [34] and the NesC programming language [26], this device, whose size is dominated by its two AA batteries, offers an 8-bit, 7.3828-MHz ATmega 128L processor, 4 kilobytes (KB) of SRAM, 128 KB of program space, 512 KB of EEPROM, and a 433-MHz radio, the baud rate of which is 38.4K and the default, per-packet payload of which (under TinyOS) is 29 KB. Such a device, given these resources, is seemingly unfit for computationally expensive or power-intensive operations. For this reason is public-key cryptography often ruled out for sensor networks as an infrastructure for authentication, integrity, privacy, and security [56, 5].

But too infrequently are such condemnations backed by actual data. In fact, save for a cursory analysis of an implementation of RSA on the MICA2 [64], little, if any, empirical research has been published on the viability of public-key infrastructure (PKI) for sensor networks.

It is precisely this void that this paper aspires to fill. By way of its own implementation of the Elliptic Curve Key Agreement Scheme, Diffie-Hellman 1 (ECKAS-DH1) [47], as well as an analysis of another's implementation of Public-Key Cryptography Standard (PKCS) #3: Diffie-Hellman Key-Agreement Standard [39], this work argues that public-key cryptography may, in fact, be tractable on the MICA2. Through instrumentation of TinyOS, it further argues that secret-key cryptography is already tractable on the same.

These arguments begin in Section 2 with an analysis of TinySec, TinyOS's existing secret-key infrastructure for the MICA2 based on SKIPJACK [17]. Section 3 redresses shortcomings in that infrastructure with a look at one implementation of Diffie-Hellman for the MICA2, based on the Discrete Logarithm Problem (DLP), meanwhile exposing weaknesses in the same. Section

4 aspires, in turn, to mitigate those weaknesses with its own implementation of Diffie-Hellman, based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), and an analysis thereof. Section 5 proposes directions for future work, while Section 6 explores related work. Section 7 concludes.

2 SKIPJACK and the MICA2

TinyOS currently offers the MICA2 access control, authentication, integrity, and confidentiality through TinySec, a link-layer security mechanism based on SKIPJACK in CBC 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 [50]. Through use of a shared, group key does TinySec provide for access control; with message authentication codes does it provide for messages' authentication and integrity; and with encryption does it provide for confidentiality.

Unfortunately, TinySec's reliance on shared keys render the mechanism particularly vulnerable to attack. 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.

But the mechanism is not without value. After all, it does offer an 80-bit key space, known attacks on which can involve up to 2^{79} operations on average (assuming SKIPJACK isn't reduced from 32 rounds [12]). And, as packets with TinySec include a 4-byte message authentication code (MAC), the probability of blind forgery is 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, as per Figure 1.

Meanwhile, the impact of TinySec on the MICA2's performance appears reasonable. On first glance, it would appear that TinySec adds under 2 milliseconds (ms) to a packet's transmission time, as per Figure 2, and under 5 ms to a packet's round-trip time (for packets echoed back to their source by some neighbor), as per Figure 3. However, the apparent overhead of TinySec, as suggested by transmission times, is nearly the data's root mean squared. Though the round-trip times exhibit less variance, additional benchmarks seem in order for TinySec's accurate analysis. Figure 4, then, offers results of yet less variance from finer instrumentation of TinySec: encryption of a 29-byte, random payload requires 2,190 μs on average, and computation of that payload's MAC requires 3,049 μs on average; overall, TinySec adds $5,239 \pm 18$ μs 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 already operating at full duty. Figure 5, the results of an analysis of the MICA2's maximal throughput, without and with TinySec enabled, puts the mechanism's computational overhead for such applications into perspective: on average, TinySec may lower maximal throughput of acknowledged packets by only 0.29 packets per second.

Of course, TinySec's encryption and authentication does come at an additional cost. Per Figure 10, TinySec adds 3,352 B collectively to an application's data and text segments, 454 B to an application's BSS 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 SRAM, then, TinySec seems a viable addition.

Unfortunately, the problem of shared keys remains. 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 shared 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 2^{32} times, that bound may be insufficient for embedded networks whose lifespans require larger IVs. Needless to say, TinySec's reliance on a single, shared key prohibits the mechanism from securely rekeying itself.

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

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

(a)

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

(b)

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

3 DLP and the MICA2

With the utility of SKIPJACK-based TinySec thus motivated and the mechanism's costs exposed, this work turns to DLP, on which Diffie-Hellman [22] is based, as the foundation for one possible answer to the MICA2's problems of shared keys' distribution and redistribution. DLP typically involves recovery of a $x \in \mathbb{Z}_p$, 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 Figure 7. Authenticated exchanges are possible with the station-to-station protocol (STS) [23], a variant of Diffie-Hellman.

Transmission Time

	without TinySec	with TinySec	Difference
Median	72,904 μs	74,367 μs	1,463 μs
Mean	74,844 μs	76,088 μs	1,244 μs
Standard Deviation	24,248 μs	24,645 μs	n/a
Standard Error	767 μs	779 μs	1,093 μs

Figure 2: Results from 1000 trials, where transmission time is defined here as the time elapsed between `SendMsg.send(·,·,·)` and `SendMsg.sendDone()` for the transmission of a 29-byte, random payload.

Round-Trip Time

	without TinySec	with TinySec	Difference
Median	145,059 μs	149,290 μs	4,231 μs
Mean	147,044 μs	152,015 μs	4,971 μs
Standard Deviation	30,736 μs	31,466 μs	n/a
Standard Error	972 μs	995 μs	1,391 μs

Figure 3: Results from 1000 trials, where round-trip time is defined here as the time elapsed between `SendMsg.send(·,·,·)` and `ReceiveMsg.receive(·)` for the transmission of a 29-byte, random payload and subsequent receipt of the same.

Computational Overhead of TinySec

	`encrypt()`	`computeMAC()`	Sum
Median	2,189 μs	3,038 μs	5,233 μs
Mean	2,190 μs	3,049 μs	5,239 μs
Standard Deviation	3 μs	281 μs	281 μs
Standard Error	0 μs	9 μs	9 μs

Figure 4: Results from 1000 trials, where `encrypt()` denotes the time required to encrypt a 29-byte, random payload, and `computeMAC()` denotes the time required to compute that payload's MAC.

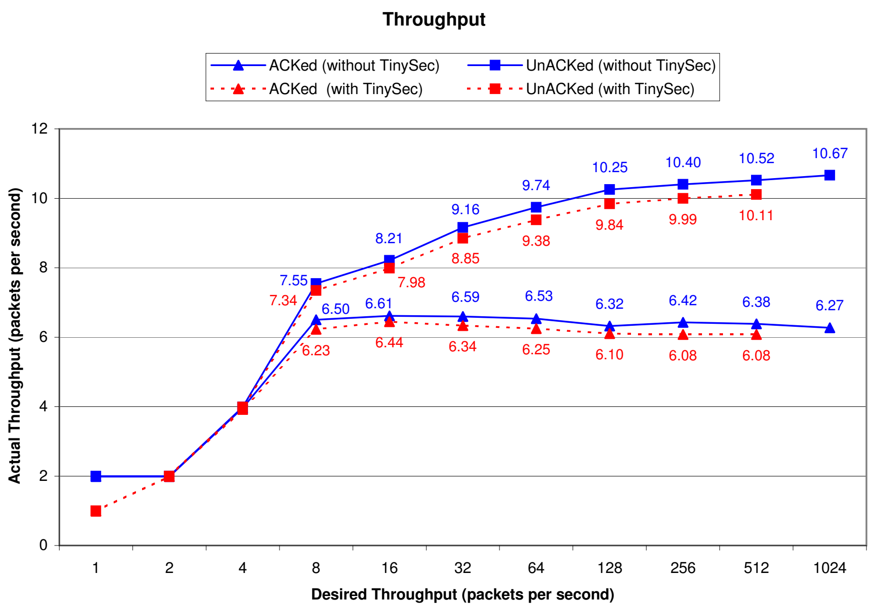


Figure 5: Actual throughput versus desired throughput for acknowledged (ACKed) and unacknowledged (unACKed) transmissions between a sender and a receiver, averaged over 1000 trials per level of desired throughput, where desired throughput is the rate at which calls to `SendMsg.send(·,·,·)` were scheduled by `Timer.start(·,·)`. ACKed actual throughput is the rate at which 29-byte, random payloads from a sender were received and subsequently acknowledged by and 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.

Memory Overhead of TinySec

	without TinySec	with TinySec	Difference
ROM	9,224 B	16,576 B	7,352 B
RAM	384 B	838 B	454 B
Stack	105 B	197 B	92 B

Figure 6: Results from instrumentation of CntToRfm, an application which simply broadcasts a counter's values over the MICA2's radio. ROM is defined here as application's data and text segments. RAM is defined here as application's BSS segment. Stack is defined here as the maximum of the application's stack size during execution.

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 bits of security, any mechanism of exchange should provide at least as much security. According to NIST, 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 [52], per Figure 8.

Unfortunately, on an 8-bit architecture, computations with 160-bit and 1,024-bit values are not inexpensive. However, modular exponentiation does not appear to be intractable on the MICA2. Figure 9 offers the results of instrumentation of one implementation of Diffie-Hellman for the MICA2 [63]: computation of $2^x \pmod{p}$, where x is a 160-bit integer and p is a well-known, 768-bit prime, requires 31.0 s on average; computation of the same, where p is a well-known, 1,024-bit prime, requires 54.9 s. Assuming (generously) that nodes sharing some key need only be rekeyed, on average, every 2^{32} packets (at which time their initialization vectors are exhausted), this computation and that for $y^x \pmod{p}$, where y is another node's public key, may be acceptable costs for an application's longevity.

Of course, these computations require that the MICA2 operate at full duty cycle, the power requirements of which may be unacceptable. After all, although the theoretical lifetime of the MICA2, powered by two AA batteries, is as many as twenty years at minimal duty cycle, that lifetime decreases to just a few days at maximal duty cycle.[1] Figure 11 reveals the power consumption of modular exponentiation on the MICA2: 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.[2]

Unfortunately, these computations 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.

But, through elliptic curve cryptography (ECC), 80 bits of security may be available to the

[1] Of course, alkaline batteries would discharge of their own accord well before 20 years.

[2] According to [16], Energizer No. E91, an AA battery, offers an average capacity of 2,850 mAh. It follows that 2 $\times$ 2,850 mAh $\times$ 3600 s/h $\div$ (7.3 mA $\times$ 54.1144 s) $\approx$ 51,945 modular exponentiations may be possible with two AA batteries on the MICA2.

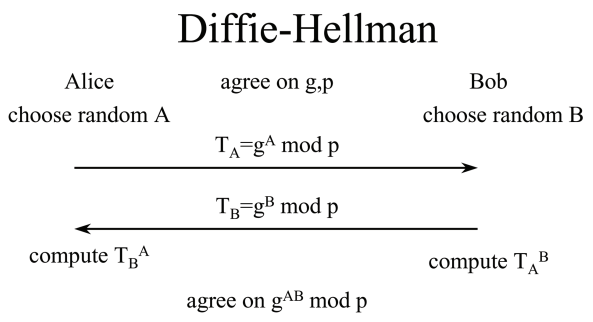


Figure 7: Typical exchange of a shared key under Diffie-Hellman based on DLP. [55]

Strength of Diffie-Hellman

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

Figure 8: Strength (*i.e.*, bits of security) of Diffie-Hellman for various moduli and exponents. [52]

MICA2 at a lower price than 1,024 bits: 163 bits. Indeed, elliptic curves are thought to offer computationally equivalent security with remarkably smaller key sizes insofar as subexponential algorithms exist for DLP [9, 27, 57, 40], but no such algorithm is known or thought to exist for ECDLP over certain fields [25, 18].

4 ECDLP and the MICA2

Elliptic curves offer an alternative foundation for the exchange of shared secrets among eavesdroppers with perfect forward secrecy, as described in Figure 13. ECDLP, on which ECC [48, 36] 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

$$y^2 + a_1xy + a_3y \equiv x^3 + a_2x^2 + a_4x + a_6, \tag{1}$$

where $a_i \in \mathbb{F}$. Of interest to cryptographers in this context have been such fields as those depicted in Figure 14. Of recent interest, however, are $\mathbb{F}_p$ and $\mathbb{F}_{2^p}$, where p is prime, as neither appears vulnerable to subexponential attack [25]. Though once popular, extension fields of composite degree over $\mathbb{F}_2$ are vulnerable by reduction with Weil descent [24] of ECDLP to DLP over hyperelliptic curves [25]. But $\mathbb{F}_{2^p}$, a binary extension field, remains popular among implementations of ECC, especially those in hardware, inasmuch 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 [54].

It is with this history in mind that I proceeded with my first, and, later, second, implementation of ECC over $\mathbb{F}_{2^p}$ toward an end of smaller public keys for the MICA2. Background for these implementations' designs now precedes their results.

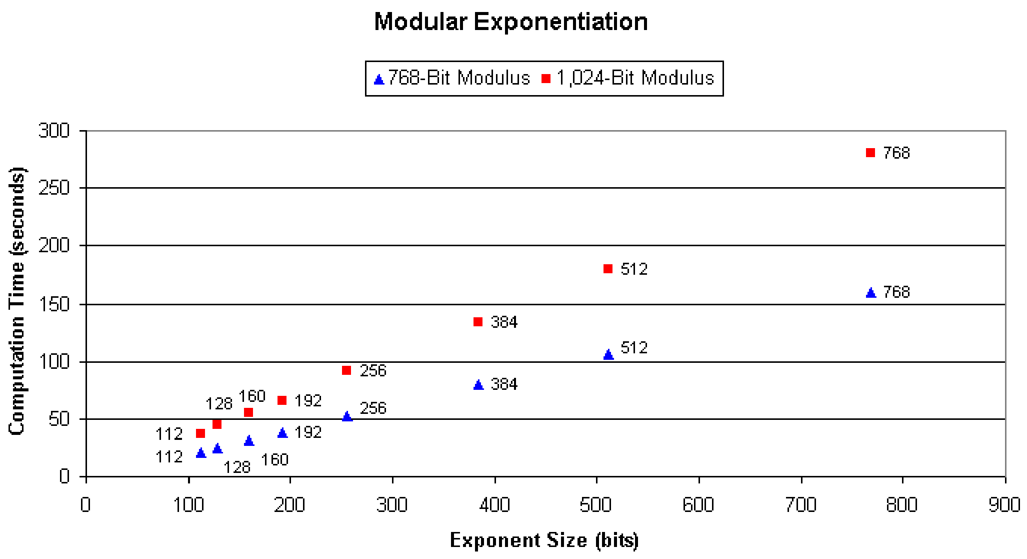


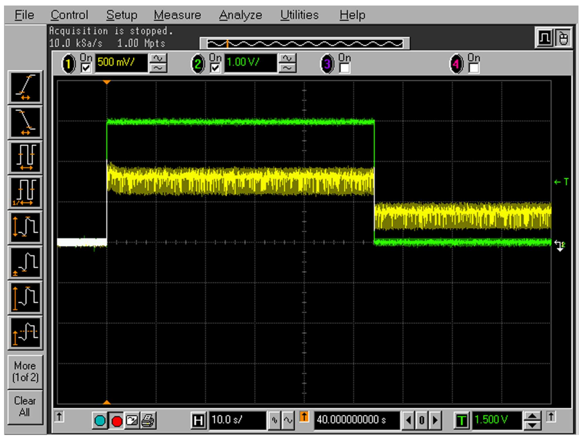
Figure 9: Time required to compute $2^x \pmod p$, where p is a well-known prime, on the MICA2.

Memory Overhead of Diffie-Hellman

	768-Bit Modulus	1,024-Bit Modulus
ROM	2,186 B	2,234 B
RAM	467 B	595 B
Stack	136 B	136 B

Figure 10: Results from instrumentation of an implementation of modular exponentiation on the MICA2 which computes $2^x \pmod p$, where x is a 512-bit integer and p is a well-known prime. ROM is defined here as application's data and text segments. RAM is defined here as application's BSS segment. Stack is defined here as the maximum of the application's stack size during execution.

Power Consumption of Modular Exponentiation



(a)

	1,024-Bit Modulus, 160-Bit Exponent
Average Current	7.3 mA
Total Time	54.1144 s
Total CPU Utilization	3.9897×10^8 cycles
Total Energy	1.185 J

(b)

Figure 11: Analyis, depicted in (a) and summarized in (b), of the power consumption of one execution of an implementation of modular exponentiation on the MICA2 which computes $2^x \pmod{p}$, where x is a 160-bit integer and p is a well-known, 1,024-bit prime. It is the yellow waveform beneath the green horizontal in (a) that represents the MICA2's relative current consumption during computation.

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, depicted in Figure 12 (albeit over $\mathbb{R}$), over $\mathbb{F}_{2^p}$ is defined as

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

where

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

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

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

where

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

where

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

provided $x_1 \neq 0$.

With these primitives is point multiplication also possible [28]. With an algebraic structure on the points of elliptic curves over $\mathbb{F}_{2^p}$ thus defined, implementation of a cryptosystem is now 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, whereby the former tends to be more efficient in software [11], though dual, triangular, and other bases exist. Admittedly, polynomial bases are also simpler conceptually and, thus, daresay, an apt choice for a first implementation of ECC on the MICA2.

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

$$a = a_{p-1}x^{p-1} + a_{p-2}x^{p-2} + \cdots + a_0x^0,$$

where, again, $a_i \in \{0, 1\}$. Accordingly, each $a \in \mathbb{F}_{2^p}$ will 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 b 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 [43].

In 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 $c(x) \equiv (\sum_{i=0}^{p-1} a_i x^i)(\sum_{i=0}^{p-1} b_i x^i) \pmod{f(x)}$.

With a polynomial basis selected, all that remains is execution of this design on the MICA2.

4.3 EccM 1.0

Version 1.0 of EccM, my first attempt at an implementation of ECC on the MICA2 in the form of a TinyOS module, is a partial success. Designed for execution on a single mote, this version of EccM first selects a random curve in the form of Equation 2, such that $a = 0$ and $b \in \mathbb{F}_{2^p}$. It next selects from that curve a random point, $G \in \mathbb{F}_{2^p} \times \mathbb{F}_{2^p}$. It further selects at random some $k \in \mathbb{F}_{2^p}$, the node's private key. Finally, it computes $k \cdot G$, the node's public key. The running time of these operations is then transmitted to the node's UART.

Based upon code by Michael Rosing [58], EccM 1.0 employs a number of optimizations. Addition of points is implemented in accordance with [59]; multiplication of points follows [37]; conversion of integers to non-adjacent form is accomplished as in [62]. Generation of pseudo-random numbers, meanwhile, is achieved with [45].

On first glance, the results, offered in Figure 15, are encouraging, with 33-bit keys requiring a running time of just 1.776 s. Unfortunately, for larger keys (*e.g.*, 63-bit), the module fails to produce results, instead causing the mote to reset cyclically. Though this behavior appears to be undocumented [19], it seems the result of stack overflow. Although none of EccM's functions are recursive, several utilize a good deal of memory for multi-word arithmetic. In fact, Figure

16 offers the results of an analysis of EccM 1.0's usage of SRAM. Unfortunately, for keys of 63 bits or more, EccM 1.0 exhausts the MICA2's SRAM.

Insofar as optimizations of EccM 1.0 fail to render generation of 63-bit keys possible, an overhaul of this first implementation proves necessary. With EccM 2.0 do I achieve my goal of 163-bit security.

4.4 EccM 2.0

Based upon the design of Dragongate Technologies Limited's Java-based jBorZoi 0.9 [42], EccM 2.0 similarly implements ECC but with greater success than EccM 1.0. Again a TinyOS module, EccM 2.0 selects for a node at random, using a polynomial basis over $\mathbb{F}_{2^p}$, a private key, thereafter computing with a curve and base point recommended by NIST the node's public key, the running time of which is then transmitted to the node's UART. In this version, multiplication of points is achieved with Algorithm IV.1 in [13]. Multiplication of elements in $\mathbb{F}_{2^p}$, meanwhile, is implemented as Algorithm 2 in [32], while inversion is implemented as Algorithm 8 in the same.

Beyond rendering 163-bit keys feasible, EccM 2.0 also redresses another shortcoming in EccM 1.0. Inasmuch as 1.0 selects curves at random, it risks (albeit with exponentially small probability) selection of supersingular curves which are vulnerable to sub-exponential attack via MOV reduction [46] with index-calculus methods [60]. EccM 2.0 thus obeys NIST's recommendation for ECC over $\mathbb{F}_{2^p}$ [51], selecting, for the results herein,

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

for the reduction polynomial,

$$y^2 + xy \equiv x^3 + x^2 + 2982236234343851336267446656627785008148015875581$$

for the curve, E, the order of (*i.e.*, number of points on) which is

$$5846006549323611672814742442876390689256843201587,$$

and, for the point $G = (G_x, G_y)$,

$$\begin{aligned} G_x &= 5759917430716753942228907521556834309477856722486 \text{ and} \\ G_y &= 1216722771297916786238928618659324865903148082417. \end{aligned}$$

Unfortunately, although EccM 2.0 employs much less memory than does EccM 1.0, per Figure 17, its running time is disappointing. The time required to generate a private and public key pair with this module, averaged over 100 trials, is 466.9 s, with a standard deviation of 16.1 s. Such performance is likely unacceptable for most applications. Moreover, the module's consumption of power is significant, as per Figure 18: generation of the node's private key requires approximately 5.7 mJ, and computation of the node's public key requires approximately 12.6402 J. In contrast with its calculation of $2^x \pmod{p}$, it appears the MICA2 could devote its lifetime to, approximately, just 4,868 of these computations.[3]

With ECC thus implemented on the MICA2 in EccM 2.0, future work nonetheless remains, the most obvious of which is reduction of this implementation's running time.

[3]This estimate follows from $2 \times 2{,}850$ mAh $\times$ 3600 s/h $\div$ (8.8 mA $\times$ 216.597 ms + 8.5 mA $\times$ 495.70 s) $\approx$ 4,868.

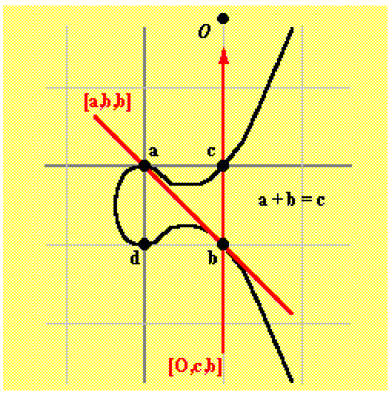


Figure 12: A representative elliptic curve (albeit of insufficient order for actual cryptography) upon which addition of $a + b = c$ is depicted. [1]

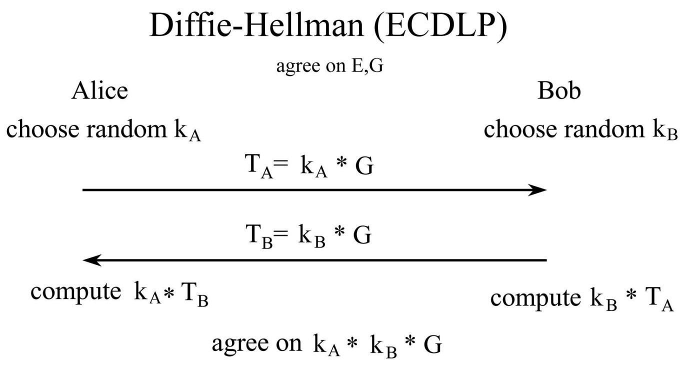


Figure 13: Typical exchange of a shared secret under Diffie-Hellman based on ECDLP.

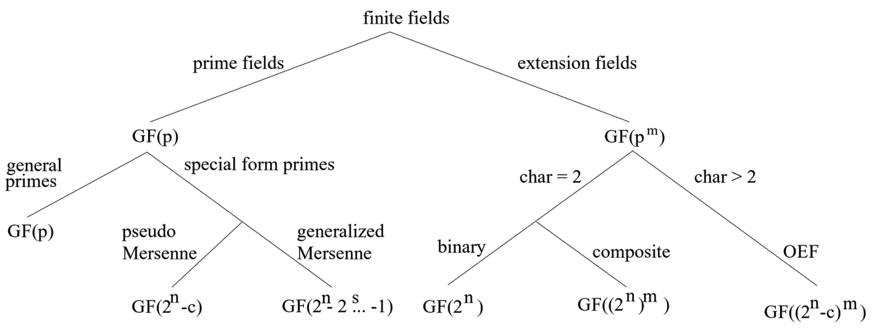


Figure 14: Finite fields proposed for use in public-key schemes. [54]

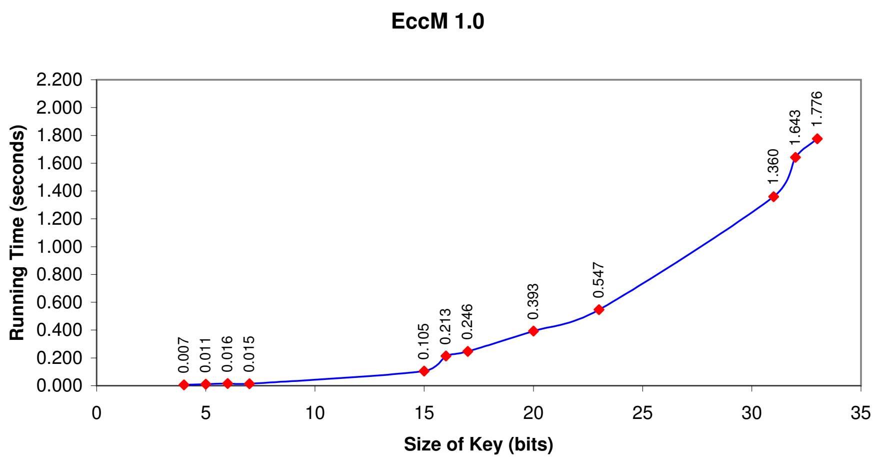


Figure 15: Running time for EccM 1.0, a TinyOS module which selects for a node at random, using a polynomial basis over $\mathbb{F}_{2^p}$, a curve, a point, and a private key, thereafter computing the node's public key, the running time of which is then transmitted to the node's UART. Points are labelled with running times. For larger keys (*e.g.*, 63-bit), the module failed to produce results.

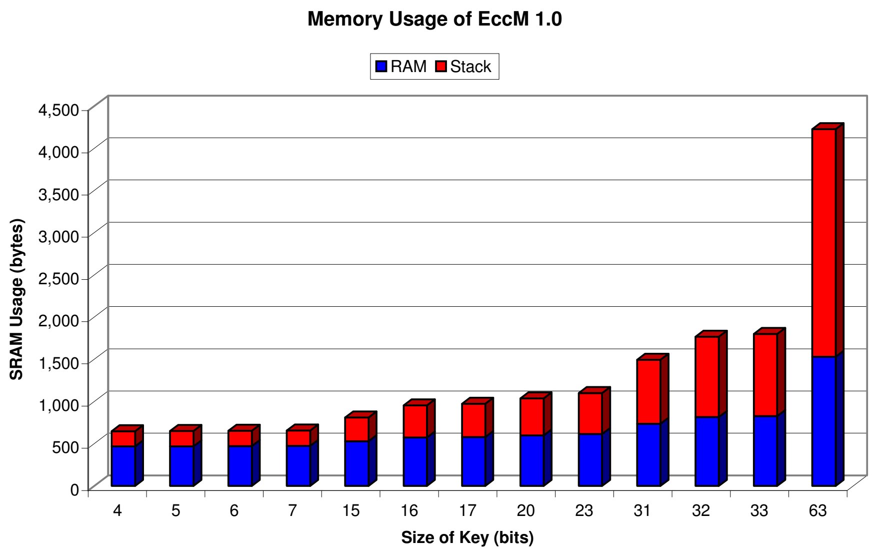


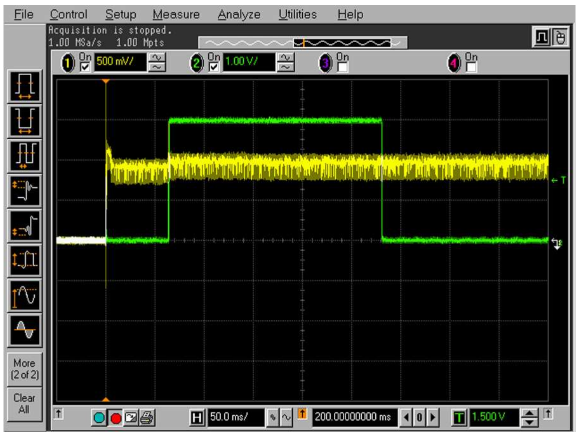
Figure 16: Analysis of EccM 1.0, a TinyOS module which selects for a node at random, using a polynomial basis over $\mathbb{F}_{2^p}$, a curve, a point, and a private key, thereafter computing the node's public key, the running time of which is then transmitted to the node's UART. RAM is defined here as application's BSS segment. 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.

Memory Usage of EccM 2.0

	163-Bit Key
ROM	43,286 B
RAM	820 B
Stack	81 B

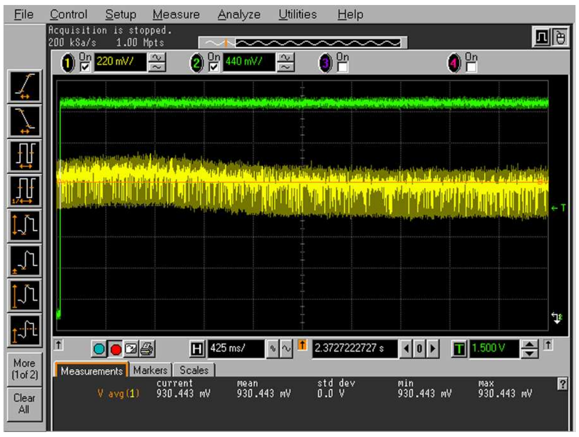
Figure 17: Analyis of EccM 2.0, a TinyOS module which selects for a node at random, using a polynomial basis over $\mathbb{F}_{2^p}$, a private key, thereafter computing with a curve and base point recommended by NIST the node's public key, the running time of which is then transmitted to the node's UART. RAM is defined here as application's BSS segment. Stack is defined here as the maximum of the application's stack size during execution.

Power Consumption of EccM 2.0



```
Acquisition is stopped.
1.00 MSa/s   1.00 Mpts
1 On 500 mV/   2 On 1.00 V/   3 On   4 On
H 50.0 ms/   200.00000000 ms   T 1.500 V
```

(a)



```
Acquisition is stopped.
200 kSa/s   1.00 Mpts
1 On 220 mV/   2 On 440 mV/   3 On   4 On
H 425 ms/   2.3727222727 s   T 1.500 V
            current      mean         std dev   min          max
V avg(1)    930.443 mV   930.443 mV   0.0 V     930.443 mV   930.443 mV
```

(b)

	Private-Key Generation	Public-Key Generation
Average Current	8.8 mA	8.5 mA
Total Time	216.597 ms	495.70 s
Total CPU Utilization	1.597×10^6 cycles	3.65×10^9 cycles
Total Energy	5.7 mJ	12.6402 J

(c)

Figure 18: Analyis, depicted in (a) and (b) and summarized in (c), of the power consumption during one execution of EccM 2.0, a TinyOS module which selects for a node at random, using a polynomial basis over $\mathbb{F}_{2^p}$, a private key, thereafter computing with a curve and base point recommended by NIST the node's public key, the running time of which is then transmitted to the node's UART. It is the yellow waveform beneath the green horizontal in (a) that represents the MICA2's relative current consumption during generation of its private key; it is the same in (b) that represents the MICA2's relative current consumption during computation of its public key.

5 Future Work

Most gratifying is 163-bit ECC's actual execution on the MICA2, but this work is but the foundation for what will hopefully become TinyCrypt, a cryptographic library for the MICA2 with support for symmetric and asymmetric keys alike.

EccM 3.0 will incorporate a number of optimizations into the source of EccM 2.0. Perhaps of greatest benefit will be 3.0's version's re-write in AVR assembly of 2.0's multi-precision arithmetic routines, to which many of the module's other functions ultimately reduce. Like C, NesC hides hardware's carry bits and overflow flags and impedes particularly efficient execution of multi-word additions, multiplications, and shifts, all of which are crucial to ECC's performance.

Although EccM 1.0 and 2.0 already incorporate a number of published algorithms, I will return to the literature for alternatives for 3.0's design.

And, perhaps more drastically, I will consider a normal basis for 3.0's arithmetic, the advantage of which is its implementation using only ANDs, XORs, and cyclic shifts, beneficiaries of which are multiplication and squaring. (For this reason do normal bases tend to be popular in implementations of ECC in hardware.) In this basis, every element $a \in \mathbb{F}_{2^p}$ can be written as $a = \sum_{i=0}^{p-1} a_i \beta^{2^i}$, where $a_i \in \{0, 1\}$. Thus defined, each such a can be represented as a binary vector, $\{a_0, a_1, \ldots, a_{p-1}\}$, where $\{\beta, \beta^2, \ldots, \beta^{2^{p-1}}\}$ is its basis over $\mathbb{F}_2$. Accordingly, each $a \in \mathbb{F}_{2^p}$ can be represented in the MICA2's SRAM as a bit string, $a_0 a_1 \cdots a_{p-1}$.

Of value to 3.0 as well might be a hybrid of polynomial and normal bases, as such is thought to leverage advantages of each simultaneously [58].

Of course, I could reconsider 3.0's choice of fields, opting instead to implement ECC over $\mathbb{F}_p$. In fact, implementation of ECC over $\mathbb{F}_p$ is relatively straightforward, as Equation 1 simplifies over prime fields to

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

where $a, b \in \mathbb{F}_p$ and $-(4a^3 + 27b^2) \neq 0$, upon substituting $x - \frac{a_2}{3}$ for x and $y - \frac{a_1 x + a_3}{2}$ for y [53]. Addition of two points, (x_1, y_1) and (x_2, y_2), is defined as

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

where

$$(x_3, y_3) \quad = \quad (\lambda^2 - x_1 - x_2,\ \lambda(x_1 - x_3) - y_1)$$

where

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

provided $x_1 \neq x_2$. Again, so that the points of the curve form an Abelian group, $\mathcal{O}$ is defined as before. And doubling of some point, (x_1, y_1), is defined is

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

where

$$(x_3, y_3) \quad = \quad (\lambda^2 - 2x_1,\ \lambda(x_1 - x_3) - y_1),$$

Recommendations for ECC

Length of Shared Key	Recommended $\|\|p\|\|$ for $\mathbb{F}_p$	Recommended p for $\mathbb{F}_{2^p}$
80	192	163
112	224	233
128	256	283
192	384	409
256	521	571

Figure 19: Recommended lengths (in bits) of private keys for ECC for exchange of shared keys of various lengths (in bits). [51]

where

$$\lambda = (3x_1^2 + a)(2y_1)^{-1},$$

provided $x_1 \neq 0$. Unfortunately, inversion, as in the above, is not inexpensive. But the operation can be avoided through use of projective (as opposed to affine) coordinates [29]. Although relatively efficient algorithms exist for modular reduction (*e.g.*, those of Montogomery [49] or Barrett [10]), 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) [61].

Contingent on its optimization, EccM 3.0 might incorporate support for larger keys, particularly those sizes in Figure 19 recommended by NIST, as well as pseudorandom generation of curves and base points in lieu of its reliance on NIST's samples.

If a success, EccM 3.0 will provide the foundation for TinyCrypt 1.0's distribution and redistribution of keys.

6 Related Work

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

Of particular relevance to this work is Woodbury's recommendation of an optimal extension field, $\mathbb{F}_{(2^8-17)^{17}}$, for low-end, 8-bit processors [66]. Jung *et al.* propose supplementary hardware for AVR implementing operations over binary fields [35]. Handschuh and Paillier propose cryptographic coprocessors for smart cards [31], whereas Woodbury *et al.* describe ECC for smart cards without coprocessors [67]. 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 [33]. Guajardo *et al.* describe an implementation of ECC for the 16-bit TI MSP430x33x family of microcontrollers [30]. Weimerskirch *et al.*, meanwhile, offer an implementation of ECC for Palm OS [65], and Brown *et al.* offer the same for Research In Motion's RIM pager [14]. ZigBee, on the other hand, shares this work's aim of wireless security for sensor networks albeit not with ECC but with AES-128 [7].

Meanwhile, recommendations for ECC's parameters abound, among academics [41], among corporations [20], and within government [51, 47].

A number of implementations of ECC in software are freely available, though none are particularly well-suited for the MICA2, in no small part due to their memory requirements. Rosing [58] offers his C-based implementation of ECC over $\mathbb{F}_{2^p}$ with both polynomial and normal bases. ECC-LIB [68] and pegwit [4] offer their own C-based implementations over $\mathbb{F}_{2^p}$ with polynomial bases. MIRACL [44] provides the same, with an additional option for curves over $\mathbb{F}_p$. LibTomCrypt [21], also in C, focuses on $\mathbb{F}_p$. Dragongate Technologies Limited, meanwhile, offers borZoi and jBorZoi [42], 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 [2].

Testament to current interest in ECC, the Workshop on Elliptic Curve Cryptography is now in its eighth year.

7 Conclusion

Despite claims to the contrary, public-key infrastructure appears viable on the MICA2. Although the implementations, studied herein, of modular exponentiation and ECC in 4 KB of primary memory on this 8-bit, 7.3828-MHz device require improvement, this work's initial results are promising indeed. AVR assembly alone offers significant potential for ECC's enhancement.

The need for PKI's success on the MICA2 seems clear. TinySec's shared keys 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 those shared keys' distribution and redistribution.

In that it offers equivalent security at lower cost to memory and bandwidth than does Diffie-Hellman based on DLP, public-key infrastructure based on elliptic curves does seem an apt choice for the MICA2. Ahead now is pursuit of this cryptosystem's minimal cost in cycles and power.

References

[1] Elliptic curve cryptography. `http://world.std.com/~dpj/elliptic.html`.

[2] libecc. `http://libecc.sourceforge.net/`.

[3] MICA2: Wireless Measurement System. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-04_A_MICA2.pdf`.

[4] pegwit. `http://groups.yahoo.com/group/pegwit/files/`.

[5] TinySec: Link Layer Security for Tiny Devices. `http://www.cs.berkeley.edu/~nks/tinysec/`.

[6] Vital Dust: Wireless Sensor Networks for Emergency Medical Care. `http://www.eecs.harvard.edu/~mdw/proj/vitaldust/`.

[7] Zigbee alliance. `http://www.zigbee.org/`.

[8] NEST Challenge Architecture. http://citeseer.nj.nec.com/gay03nesc.html, August 2002.

[9] L. M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proc. 20th IEEE Found. Comp. Sci. Symp.*, pages 55–60, 1979.

[10] P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263, 1987.

[11] George Barwood. Elliptic curve cryptography FAQ v1.12 22nd. `http://www.cryptoman.com/elliptic.htm`, December 1997.

[12] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. *Lecture Notes in Computer Science*, 1592:12–23, 1999.

[13] I. Blake, G. Seroussi, and N. Smart. Elliptic curves in cryptography. *LMS Lecture Note Series*, 265, 1999.

[14] Michael Brown, Donny Cheung, Darrel Hankerson, Julio Lopez Hernandez, Michael Kirkup, and Alfred Menezes. Pgp in constrained wireless devices. In *Proceedings of the 9th USENIX Security Symposium*. USENIX Association, August 2000.

[15] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology, 2001.

[16] Everyready Battery Company. Engineering datasheet: Energizer no. x91. `http://data.energizer.com/datasheets/library/primary/alkaline/energizer/consumer_oem/e91.pdf`.

[17] Computer Security Division. *SKIPJACK and KEA Algorithm Specifications*. National Institute of Standards and Technology, May 1988.

[18] Certicom Corp. Remarks on the security of the elliptic curve cryptosystem. `http://www.comms.engg.susx.ac.uk/fft/crypto/EccWhite3.pdf`, July 2000.

[19] Atmel Corporation. *ATmega128(L) Preliminary Complete*. San Jose, CA, December 2003.

[20] Certicom Corporation. Standards for efficient cryptography group. `http://www.secg.org/`.

[21] Tom St Denis. LibTomCrypt. `http://libtomcrypt.org/`.

[22] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[23] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2(2):107–125, 1992.

[24] Gerhard Frey and Herbert Gangl. How to disguise an elliptic curve (weil descent). ECC '98, September 1998.

[25] P. Gaudry, F. Hess, and N. P. Smart. Constructive and destructive facets of weil descent on elliptic curves. Technical Report CSTR-00-016, Department of Computer Science, University of Bristol, October 2000.

[26] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems.

[27] Daniel M. Gordon. Discrete logarithms in GF(P) using the number field sieve. *SIAM J. Discret. Math.*, 6(1):124–138, 1993.

[28] Daniel M. Gordon. A survey of fast exponentiation methods. *J. Algorithms*, 27(1):129–146, 1998.

[29] Johann Groβschädl. Implementation options for elliptic curve cryptography. `http://www.iaik.tugraz.at/teaching/02_it-sicherheit/04_vortragsthemen/ECC.pdf`, April 2003.

[30] Jorge Guajardo, Rainer Blümel, Uwe Krieger, and Christof Paar. Efficient implementation of elliptic curve cryptosystems on the ti msp430x33x family of microcontrollers. In K. Kim, editor, *PKC 2001*, pages 365–382, Korea, 2001.

[31] Helena Handschuh and Pascal Paillier. Smart card crypto-coprocessors for public-key cryptography. In J.-J. Quisquater and B. Schneier, editors, *Lecture Notes in Computer Science*, Smart Card Research and Applications, pages 386–394. Springer-Verlag, 2000.

[32] Darrel Hankerson, Julio López Hernandez, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. *Lecture Notes in Computer Science*, 1965, 2001.

[33] Toshio Hasegawa, Junko Nakajima, and Mitsuru Matsui. A Small and Fast Software Implementation of Elliptic Curve Cryptosystems over GF(p) on a 16-Bit Microcomputer. *IEICE Trans. Fundamentals*, E82-A(1):98–106, January 1999.

[34] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[35] Michael Jung, M. Ernst, F. Madlener, S. Huss, and R. Blümel. A reconfigurable system on chip implementation for elliptic curve cryptography over gf(2^n). `http://ece.gmu.edu/crypto/ches02/talks_files/Jung.ppt`.

[36] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[37] N. Koblitz. Cm-curves with good cryptographic properties. In *Advances in Cryptology – CRYPTO '91*, pages 279–287, 1992.

[38] Venkata A. Kottapalli, Anne S. Kiremidjian, Jerome P. Lynch, Ed Carryer, Thomas W. Kenny, Kincho H. Law, and Ying Lei. Two-tiered wireless sensor network architecture for structural health monitoring, March 2003.

[39] RSA Laboratories. Lightweight Security for Wireless Networks of Embedded Systems. `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-3/`, November 1993.

[40] B. A. LaMacchia and A. M. Odlyzko. Computation of discrete logarithms in prime fields. *Lecture Notes in Computer Science*, 537:616–618, 1991.

[41] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. October 1999.

[42] Dragongate Technologies Limited. jborzoi 0.9. `http://dragongate-technologies.com/products.html`, August 2003.

[43] Julio López and Ricardo Dahab. An overview of elliptic curve cryptography. Technical report, Institute of Computing, Sate University of Campinas, São Paulo, Brazil, May 2000.

[44] Shamus Software Ltd. Multiprecision integer and rational arithmetic c/c++ library. `http://indigo.ie/~mscott/#Elliptic`.

[45] George Marsaglia. The mother of all random generators. `ftp://ftp.taygeta.com/pub/c/mother.c`, October 1994.

[46] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 80–89. ACM Press, 1991.

[47] Microprocessor and Microcomputer Standards Committee. *IEEE Standard Specifications for Public-Key Cryptography*. IEEE Computer Society, January 2000.

[48] V. Miller. Uses of elliptic curves in cryptography. In *Lecture Notes in Computer Science 218: Advances in Crytology - CRYPTO '85*, pages 417–426. Springer-Verlag, Berlin, 1986.

[49] Peter Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

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

[51] National Institute of Standards and Technology. Recommended elliptic curves for federal government use. `http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf`, July 1999.

[52] National Institute of Standards and Technology. Special Publication 800-57: Recommendation for Key Management, January 2003.

[53] Elisabeth Oswald. Introduction to elliptic curve cryptography. `http://www.iaik.tugraz.at/aboutus/people/oswald/papers/Introduction_to_ECC.pdf`, July 2002.

[54] C. Paar. Implementation options for finite field arithmetic for elliptic curve cryptosystems. Invited presentation at the 3rd Workshop on Elliptic Curve Cryptography (ECC '99), November 1999.

[55] Radia Perlman. Computer Science 243.

[56] Adrian Perrig, Robert Szewczyk, Victor Wen, David E. Culler, and J. D. Tygar. SPINS: security protocols for sensor networks. In *Mobile Computing and Networking*, pages 189–199, 2001.

[57] M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT, 1979.

[58] Michael Rosing. *Implementing Elliptic Curve Cryptography.* Manning Publications Co., 1999.

[59] Richard Schroeppel, Hilarie Orman, Sean O'Malley, and Oliver Spatscheck. Fast key exchange with elliptic curve systems. *Lecture Notes in Computer Science*, 963, 1995.

[60] Silverman and Suzuki. 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, 1998.

[61] J. Solinas. Generalized mersenne numbers. Technical Report CORR-39, University of Waterloo, 1999.

[62] J. A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In *Advances in Cryptology – CRYPTO '97*, pages 357–371, 1997.

[63] BBN Technologies. Diffie-hellman 1, July 2003.

[64] Ronald Watro. Lightweight Security for Wireless Networks of Embedded Systems. `http://www.is.bbn.com/projects/lws-nest/bbn_nest_apr_03.ppt`, May 2003.

[65] André Weimerskirch, Christof Paar, and Sheueling Chang Shantz. Elliptic curve cryptography on a palm os device. Sydney, Australia, July 2001. The 6th Australasian Conference on Information Security and Privacy.

[66] Adam D. Woodbury. Efficient algorithms for elliptic curve cryptosystems on embedded systems. `http://www.wpi.edu/Pubs/ETD/Available/etd-1001101-195321/unrestricted/woodbury.pdf`, September 2001.

[67] Adam D. Woodbury, Daniel V. Bailey, and Christof Paar. Elliptic curve cryptography on smart cards without coprocessors. Bristol, UK, September 2000. The Fourth Smart Card Research and Advanced Applications (CARDIS 2000) Conference.

[68] Christos Zaroliagis. ECC-LIB: A Library for Elliptic Curve Cryptography. `http://www.ceid.upatras.gr/faculty/zaro/software/ecc-lib/`.