Energy-Efficient Implementation of ECDH Key Exchange for Wireless Sensor Networks

Christian Lederer[1], Roland Mader[2,3], Manuel Koschuch[4], Johann Großschädl[5], Alexander Szekely[6], and Stefan Tillich[6]

[1] University of Klagenfurt, Austria
christian.lederer@uni-klu.ac.at
[2] ITI, Graz University of Technology, Austria
roland.mader@tugraz.at
[3] AVL List GmbH, Austria
roland.mader@avl.com
[4] FH Campus Wien – University of Applied Sciences, Austria
manuel.koschuch@fh-campuswien.ac.at
[5] University of Bristol, United Kingdom
johann.groszschaedl@cs.bris.ac.uk
[6] IAIK, Graz University of Technology, Austria
{aszekely,stillich}@iaik.tugraz.at

Abstract. Wireless Sensor Networks (WSNs) are playing a vital role in an ever-growing number of applications ranging from environmental surveillance over medical monitoring to home automation. Since WSNs are often deployed in unattended or even hostile environments, they can be subject to various malicious attacks, including the manipulation and capture of nodes. The establishment of a shared secret key between two or more individual nodes is one of the most important security services needed to guarantee the proper functioning of a sensor network. Despite some recent advances in this field, the efficient implementation of cryptographic key establishment for WSNs remains a challenge due to the resource constraints of small sensor nodes such as the MICAz mote. In this paper we present a lightweight implementation of the elliptic curve Diffie-Hellman (ECDH) key exchange for ZigBee-compliant sensor nodes equipped with an ATmega128 processor running the TinyOS operating system. Our implementation uses a 192-bit prime field specified by the NIST as underlying algebraic structure and requires only $5.20 \cdot 10^6$ clock cycles to compute a scalar multiplication if the base point is fixed and known a priori. A scalar multiplication using a random base point takes about $12.33 \cdot 10^6$ cycles. Our results show that a full ECDH key exchange between two MICAz motes consumes an energy of 57.33 mJ (including radio communication), which is significantly better than most previously reported ECDH implementations on comparable platforms.

1 Introduction

A Wireless Sensor Network (WSN) is a network consisting of a (potentially very large) number of autonomous devices, so-called motes, which are deployed in

O. Markowitch et al. (Eds.): WISTP 2009, LNCS 5746, pp. 112–127, 2009.

the environment to cooperatively monitor physical conditions like temperature [37]. The sensor nodes are equipped with radio transceivers, enabling them to communicate with other nodes and centralized resources (e.g. a base station) or to connect to the Internet. In fact, WSNs are a prime example of what is often referred to by such buzz phrases as "pervasive computing," "smart dust," or the "internet of things" [10]. The February 2003 issue of the magazine Technology Review listed WSNs among 10 emerging technologies that will change the world [6]. Today, WSNs play a vital role in a multitude of applications ranging from environmental surveillance over medical monitoring to home automation [37]. A recent study [32] predicts that the WSN market for smart homes will grow from $470 million in 2007 to up to $2.8 billion in 2012, with a potential market size of 6 billion cumulative sensor nodes worldwide.

Security and privacy issues pose a big challenge for the widespread adoption of WSN technology in certain application domains such as health care, traffic control, or disaster detection [7,28]. Unfortunately, WSNs are easier to attack (and harder to protect) than other types of network like, for example, corporate intranets. There are basically three reasons why unprotected WSNs are an easy target for malicious attacks. First, the wireless communication between nodes via radio signals makes eavesdropping quite easy and facilitates a slew of active attacks ranging from message injection to denial of service [12]. Second, the deployment of WSNs in unattended areas gives an attacker direct access to the sensor nodes and enables him to conduct all kinds of physical attacks including node capture [3]. Third, the vast majority of sensor nodes on the market today are battery-operated and, hence, severely restricted in terms of computational power, which makes the implementation of cryptographic schemes and security protocols rather difficult. To save energy, WSN designers often refrain from an attempt to secure the network or implement "futile" security measures like the encryption of node-to-node communication using a single network-wide key.

1.1 Key Establishment in WSNs

The establishment of a secret key shared between two (or more) sensor nodes is undoubtedly one of the most important security services needed to ensure the integrity and well functioning of a WSN. Various key establishment techniques taking the special characteristics and adversary models of sensor networks into account have been proposed [27,42]. A simple yet effective approach to obtain shared secret keys in a WSN is *random key pre-distribution*, first introduced by Eschenauer and Gligor in [16]. The idea is to load a set of keys randomly chosen from a large key pool onto each node prior to deployment such that two nodes will share (at least) one key with a certain probability. While this basic scheme is easy to implement and entails only little overhead since no costly key agreement must be carried out, it has some disadvantages in terms of scalability and resilience to node capture. Several improvements of Eschenauer's probabilistic key pre-distribution scheme have been published, including a variant where two sensor nodes must share $q > 1$ common keys instead of just a single one [9]. The benefit of this increased amount of key overlap is better resilience against node

capture. Another variant described in [9] supports node-to-node authentication by assigning a unique secret key to each pair of nodes. Liu and Ning proposed in [26,27] a polynomial pool-based key pre-distribution scheme which combines probabilistic key pre-distribution and polynomial-based key generation to obtain a shared secret. In this scheme, the key pool is replaced by a pool of randomly generated bivariate polynomials over a finite field, and each node is pre-loaded with a set of polynomial shares (i.e. partially evaluated polynomials). Two nodes possessing polynomial shares of the same bivariate polynomial can establish a secret key following the *polynomial-based key distribution* protocol described in [5]. A very similar key establishment technique was published in [15] along with an in-depth theoretical analysis of its security properties. The polynomial pool-based scheme features low communication overhead and is substantially more resilient against node capture than the basic Eschenauer-Gligor scheme and the q-composite scheme from [9] as long as the number of compromised nodes does not exceed a certain threshold. Zhu et al presented in [44] a scalable protocol for key establishment based on the ideas of probabilistic key sharing and threshold secret sharing. The resilience of this protocol remains intact under a collusion attack by up to a certain number of compromised nodes, similar to the Liu-Ning scheme. Another common feature of the schemes in [26,15] and [44] is that they enable a pair of nodes to set up a unique secret key exclusively known by these two nodes. Therefore, compromised nodes will not leak information about the secret keys shared among non-compromised nodes[7].

A completely different approach for key establishment in WSNs is to use a trusted third party (e.g. the base station) that acts as a *key distribution center (KDC)* and generates, upon request, a unique secret key for two nodes wishing to communicate securely with each other. The KDC sends this key in encrypted form to the two sensor nodes, similar to the Needham-Schroeder protocol [31] or *Kerberos* [24]. Kerberos was originally designed to authenticate entities on a network; the establishment of a secret key is a "side effect." Each node of the network shares a long-term secret key with the KDC, which enables the nodes to verify that messages from the KDC are authentic. Similarly, knowledge of the long-term key also serves to prove a node's identity to the KDC. To set up a link key shared between node A and node B, the KDC generates a secret key and securely sends it to A and B encrypted under the long-term key it shares with A and with B, respectively. Extraction of the link key is only possible for the legitimate node which possesses the corresponding long-term key. Thus, by trusting the KDC, the nodes can authenticate each other (i.e. prove their true identity) and establish a secret key. The long-term key that each sensor node

[7] Per definition, a *pair-wise key establishment scheme* assigns a unique secret key to each pair of nodes. The polynomial pool-based key pre-distribution scheme [26,15] can fulfill this property, provided that no polynomial of degree t is used more than $t+1$ times. Zhu et al's scheme [44] is strictly speaking not pair-wise, but guarantees with an overwhelming probability that a secret key is exclusively known to a pair of nodes. On the other hand, the basic Eschenauer-Gligor scheme is not a pair-wise scheme since one and the same key may be used by several node pairs.

shares with the KDC is pre-deployed, but, contrary to approaches with a single network-wide key, each node has a unique key. Therefore, compromised nodes do not jeopardize the security of the rest of the WSN, which makes Kerberos-like protocols very robust against node capture. However, they suffer from high communication cost, especially in large networks in which the base station may be located far away from the two nodes wishing to set up a link key. A second drawback of protocols using a central KDC is their non-uniform communication pattern: Nodes located in the vicinity of the KDC have to forward all requests for link keys from the rest of the WSN, which drains the batteries of these nodes at a high rate. The concentration of network traffic near the KDC clearly limits the scalability of Kerberos. To alleviate these disadvantages, Chan and Perrig [8] introduce PIKE, a key establishment protocol that uses "ordinary" nodes as trusted intermediaries for the generation of link keys. Both the communication cost and memory overhead of PIKE scale with $\mathcal{O}(\sqrt{n})$, where n represents the number of nodes in the network. Perrig et al describe in [33] a Kerberos-like key establishment technique implemented on top of the Secure Network Encryption Protocol (SNEP).

Key establishment in WSNs can also be performed with protocols that use *public-key cryptography* to generate a secret key shared between two nodes. The most important key exchange protocol was proposed by Diffie and Hellman [14] in 1976 and is usually implemented in the multiplicative group of a finite field of prime order. Alternatively, it is also possible to embed the Diffie-Hellman key exchange into an additive group like the group of points on an elliptic curve defined over a finite field. The efficient implementation of *elliptic curve cryptography (ECC)* for WSN has been an active area of research in recent years, in particular since Gura et al [21] demonstrated that Elliptic Curve Diffie-Hellman (ECDH) key exchange is feasible for resource-restricted sensor nodes. The main advantages of using ECDH key exchange in WSNs are perfect resilience to node capture, excellent scalability, and low memory as well as communication overhead. However, the big drawback of ECDH is the highly computation-intensive nature of its underlying cryptographic operations, causing long execution times and high energy consumption. Energy is the most precious resource of wireless motes, and this will remain so for the next future since dramatic improvements in battery technology are not foreseen. Therefore, approaches for reducing the energy cost of ECDH key exchange are eagerly sought.

1.2 Our Contributions

In this paper we present a highly-optimized software implementation of ECDH key exchange for ZigBee-compliant sensor nodes running the TinyOS operating system, in particular the MICAz motes [11]. Contrary to previous work, where in most cases an elliptic curve over a 160-bit prime field was used as underlying algebraic structure, we base our implementation on a cryptographically much stronger curve over a 192-bit field that is compliant with all major standards for ECC, including the NIST recommendations [30]. We integrated our ECC code into an experimental TinyOS program for key exchange between two MICAz

motes, which allowed us to conduct a detailed performance and energy analysis of both the cryptographic operations and the radio communication. Our work advances the state-of-the-art in efficient ECDH implementation in the following ways: First, we present an improved version of Gura et al's [21] hybrid method for multi-precision multiplication that requires fewer single-precision additions (i.e. `add` and `adc` instructions on an ATmega128L processor [2]). Our variant is similar (but not identical) to the hybrid multiplication methods introduced in [35] and [40]. Second, our implementation uses fast algorithms for elliptic curve scalar multiplication (window method, comb method) to reduce the execution time of ECDH key exchange at the expense of a slight increase in memory requirements. However, we show that despite the additional memory demand, the window and comb methods are perfectly feasible for MICAz motes. Third, we aimed to secure our ECDH implementation against side-channel attacks. Thanks to the window and comb methods, the performance degradation caused by the integration of side-channel countermeasures is relatively small.

2 Elliptic Curve Cryptography

An elliptic curve E over a prime field $\mathbb{F}_p$ can be defined as the set of all tuples $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ satisfying an equation of the form

$$y^2 = x^3 + ax + b \quad \text{with} \quad a, b \in \mathbb{F}_p \tag{1}$$

These tuples are called *points* with x and y referred to as coordinates. The set of points together with a special point $\mathcal{O}$ (the so-called *point at infinity*) allows one to form a commutative group with $\mathcal{O}$ being the identity element. The group operation is the addition of points, which can be performed through arithmetic operations (addition, subtraction, multiplication, squaring, and inversion) in the underlying field $\mathbb{F}_p$ according to well-defined formulae (see e.g. [22]). Adding a point $P = (x, y)$ to itself is referred to as point doubling and can also be done through a well-defined sequence of operations in $\mathbb{F}_p$. In general, point doubling requires fewer field operations than the addition of two points.

The *order* of an elliptic curve group $E(\mathbb{F}_p)$ is the number of $\mathbb{F}_p$-rational points on the curve E, plus one for the point at infinity. It is well known from Hasse's theorem that $\#E(\mathbb{F}_p)$ has the following bounds:

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p} \tag{2}$$

For cryptographic applications, $\#E(\mathbb{F}_p)$ should have a large prime factor; in the ideal case it is a prime. Before ECDH key exchange (or any other elliptic curve scheme) can be carried out, the involved parties have to agree upon a common set of so-called *domain parameters*, which specify the finite field $\mathbb{F}_p$, the elliptic curve E (i.e. the coefficients $a, b \in \mathbb{F}_p$ defining E according to Equation (1)), a base point $P \in E(\mathbb{F}_p)$ generating a cyclic subgroup of large order, the order n of this subgroup, and the co-factor $h = \#E(\mathbb{F}_p)/n$. Consequently, elliptic curve domain parameters over $\mathbb{F}_p$ are simply a sextuple $D = (p, a, b, P, n, h)$ [22]. In

elliptic curve cryptography, a private key is an integer k chosen randomly from the interval $[1, n-1]$. The corresponding public key is the point $Q = k \cdot P$ on the curve. Given k and P, the point $Q = k \cdot P$ can be obtained by means of an operation called *scalar multiplication* [22]. Numerous algorithms for scalar multiplication have been proposed; the simplest way to compute $k \cdot P$ is to perform a sequence of point additions and doublings, similar to the square-and-multiply algorithm for modular exponentiation.

While a scalar multiplication of the form $Q = k \cdot P$ can be calculated quite efficiently, the inverse operation, i.e. finding k when P and Q are given, is a hard mathematical problem known as the *Elliptic Curve Discrete Logarithm Problem (ECDLP)*. To date, the best algorithm known for solving the ECDLP requires fully exponential time if the domain parameters were chosen with care [22]. In contrast, the best algorithm for solving the Discrete Logarithm Problem (DLP) in $\mathbb{Z}_p^*$ or the Integer Factorization Problem (IFP) has a sub-exponential running time. As a consequence, elliptic curve cryptosystems can use much shorter keys compared to their "classical" counterparts based on the DLP or IFP. A common rule of thumb states that a properly designed 160-bit ECC scheme is about as secure as 1024-bit RSA. However, the U.S. National Institute of Standards and Technology (NIST) recommends using 1024-bit RSA and 160-bit ECC only until 2010. Therefore, we opted to embed our implementation of the ECDH protocol into a much stronger 192-bit elliptic curve group.

2.1 Elliptic Curve Diffie-Hellman (ECDH) Key Exchange

ECDH key exchange is the elliptic curve analogue of the classical Diffie-Hellman key exchange operating in $\mathbb{Z}_p^*$ [14]. As its classical counterpart, the ECDH protocol can be used to establish a shared secret key between two entities using an insecure communication channel. In the following, we describe in detail the steps that two communicating parties, usually called Alice and Bob, have to perform in order to obtain a shared secret. We assume that Alice and Bob use the same set of domain parameters $D = (p, a, b, P, n, h)$ for their computations.

- Alice generates an ephemeral key pair (k_A, Q_A), i.e. she generates a random number k_A in the interval $[1, n-1]$ and then performs a scalar multiplication to get the corresponding public key $Q_A = k_A \cdot P$. She sends Q_A to Bob.
- Bob generates an ephemeral key pair (k_B, Q_B) with $Q_B = k_B \cdot P$ in the same way as described above and sends the public key Q_B to Alice.
- After Alice receives Bob's ephemeral public key Q_B, she performs a scalar multiplication to obtain the shared secret $S = k_A \cdot Q_B$.
- After Bob receives the ephemeral public key Q_A from Alice, he obtains the shared secret through computation of $S = k_B \cdot Q_A$.

Now Alice and Bob possess the same secret S since $k_A \cdot Q_B = k_A \cdot k_B \cdot P$ and $k_B \cdot Q_A = k_B \cdot k_A \cdot P$, i.e. both parties arrived at the same value for S because $E(\mathbb{F}_p)$ is a commutative group. Each run of the ECDH protocol requires Alice and Bob to send two messages (to exchange the ephemeral public keys) and to

perform four scalar multiplications altogether. The first two could be computed simultaneously by Alice and Bob; the other two scalar multiplications must be carried out thereafter. It is also possible to precalculate a pair of ephemeral keys when the parties are idling to speed up subsequent protocol runs.

An attacker might intercept the public keys Q_A and Q_B, but he will not be able to derive the private keys k_A and k_B from Q_A, Q_B, P unless he solves the ECDLP. The security of the ECDH protocol relies on the intractability of the (computational) Elliptic Curve Diffie-Hellman Problem (ECDHP); that is, given an elliptic curve E, a base point $P \in E(\mathbb{F}_p)$, and two points $Q_A = k_A \cdot P$ and $Q_B = k_B \cdot P$, find the point $S = k_A \cdot k_B \cdot P$ without knowledge of k_A, k_B. It is clear that an algorithm for solving a generic ECDLP instance would allow one to solve the ECDHP as well.

A straightforward implementation of ECDH key exchange is vulnerable to a man-in-the-middle attack [22]. To prevent this attack, the ECDH protocol as described above must be extended in such a way that the communicating parties are authenticated to each other. Nonetheless, the classical ECDH key exchange serves as a good benchmark for the feasibility of public-key cryptography on resource-constrained sensor nodes. Key exchange in WSNs using an advanced protocol incorporating authentication, such as Signed ECDH or ECMQV, has been studied in [13] and [20], respectively.

2.2 Scalar Multiplication

The computationally expensive part of virtually all elliptic curve cryptosystems is scalar multiplication, an operation of the form $k \cdot P$ where k is an integer and P is a point on the curve. A scalar multiplication can be performed by means of repeated point additions and point doublings, both of which, in turn, involve a sequence of arithmetic operations (i.e. addition, multiplication, squaring, and inversion) in the underlying finite field. Inversion is by far the most expensive operation in prime fields [22]. However, it is possible to add points on an elliptic curve without the need to perform costly inversions, e.g. by representing the points in *projective coordinates* [22]. In Section 2 we described the conventional (i.e. affine) coordinate system in which a point P is associated with an x and a y coordinate, i.e. a tuple $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$. By contrast, in projective coordinate systems, a point is represented by a triplet (X, Y, Z), which corresponds to the affine coordinates $(X/Z^u, Y/Z^v)$ when $Z \neq 0$ (u and v depend on the specific coordinate system chosen). For example, the projective point $P = (X, Y, Z)$ in Jacobian coordinates corresponds to the affine point $P = (X/Z^2, Y/Z^3)$. It is also possible to add two points when one is given in projective coordinates and the other in affine coordinates [22]. In fact, such mixed coordinates often lead to very efficient point addition formulae. For example, adding a point in Jacobian coordinates to an affine point requires eight multiplications and three squarings in $\mathbb{F}_p$ (but no inversion). Doubling a point given in Jacobian coordinates takes four multiplications and four squarings.

The *double-and-add* algorithm performs a scalar multiplication via repeated point additions and doublings, analogous to the multiply-and-square algorithm

for modular exponentiation. It uses the binary expansion of the integer k and computes $k \cdot P$ as follows: For each bit k_i of k, the current intermediate result is doubled, and the base point P is added if bit $k_i = 1$ (no addition is performed when $k_i = 0$). Given an l-bit scalar k, the double-and-add algorithm executes exactly l point doublings, whereas the number of point additions depends on the Hamming weight of k. In the average case $l/2$ additions are carried out; the worst-case number of additions is l. The conventional double-and-add method can be easily improved by using a signed-digit representation of k. One option is the *non-adjacent form (NAF)*, which reduces the number of additions (of either P or $-P$) to $l/3$ in the average case and $l/2$ in the worst case [22]. However, the number of point doublings remains the same.

The average number of point additions can be further reduced if some RAM is available for storing multiples of the base point P. A *window method* with a window size of w uses a radix-2^w representation of the scalar k and requires to pre-compute the points $2P, 3P, \ldots, (2^w - 1)P$. These $2^w - 2$ points are stored in a look-up table, typically in affine representation to save RAM and to allow one using mixed coordinates for point addition. The window method works in a similar fashion as the double-and-add method, except that in each step w bits of k are considered with the corresponding table-entry being added to the intermediate result. A window size of w reduces the total number of point additions to roughly l/w, but does not change the number of doublings. Results from previous work show that $w = 4$ represents a good compromise between performance and memory requirements.

If the base point is fixed an known a priori, which is the case when generating an ephemeral key pair for ECDH key exchange, the number of both additions and doublings can be reduced by using a so-called *comb method* [22]. The idea is to pre-compute the points $P_i = 2^{wi} \cdot P$ for $0 \leq i \leq l/w - 1$ and to perform the scalar multiplication in an interleaved fashion (similar to Shamir's trick), which yields a total of l/w doublings and roughly the same number of additions. As the base point P is fixed, it is possible to do the pre-computation off-line and store a look-up table holding the $2^w - 2$ points in Flash memory or ROM. The window method and the comb method have in common that the average-case and the worst-case execution time are almost the same.

3 Prime-Field Arithmetic on the ATmega128

In this section we describe the implementation and optimization of prime-field arithmetic on MICAz motes from Crossbow Technologies [11]. The MICAz is a low-power sensor node equipped with an 8-bit ATmega128L processor clocked at 7.3728 MHz, 4 kB RAM, and 128 kB Flash memory. It also features an IEEE 802.15.4 ("ZigBee") compliant RF transceiver, which allows for communication with other nodes and the base station. The ATmega128 is a simple 8-bit RISC processor [2] based on the AVR instruction set [1], i.e. the usual arithmetic and logical instructions are supported, including a fast integer-multiply instruction with a 2-cycle latency. A total of 32 general-purpose registers are available.

Our implementation of the ECDH protocol uses a NIST-recommended elliptic curve over a 192-bit prime field as basic building block. The field is defined by the generalized-Mersenne prime $p = 2^{192} - 2^{64} - 1$ [30]. All arithmetic operations described in this section are performed on (and optimized for) 192-bit operands (i.e. 192-bit integers). It is common practice in multiple-precision arithmetic to store the operands in arrays of single-precision words, e.g. arrays of unsigned m-bit integers with m denoting the processor's word size. However, the ANSI C standard specifies the size of the basic integer type to be at least 16 bits, even on 8-bit platforms. Therefore, we decided to use a 16-bit representation, i.e. a 192-bit field element is stored in an array of $s = 12$ words, each accommodating 16 bits. All software routines were designed and implemented with 16-bit words as the "smallest unit" of data, which means they operate on two bytes of the operand(s) at a time. Another important characteristic of our implementation is that we tolerate incompletely reduced results, provided that their length does not exceed 192 bits. In other words, an operand does not necessarily need to be in the interval $[0, p-1]$, but it must be smaller than 2^{192} so that it fits into a 12-word array.

3.1 Addition and Subtraction

The addition of two field elements a, b is implemented via a loop that iterates through the words of the operands, starting with the least significant word. In each iteration, a word (i.e. two bytes) of operand a and a word of b are loaded from memory and added up using the `add` (resp. `adc`) instruction. After addition of the most significant word, the prime p must be subtracted if the sum exceeds 192 bits, which can be easily checked via the carry flag. Note that we tolerate an incompletely reduced result; thus it is not necessary to do an exact comparison between the sum and p. The field subtraction is implemented as conventional subtraction, followed by an addition of p if the result was negative.

3.2 Multiplication and Squaring

The overall execution time of a scalar multiplication depends significantly on the efficiency of the multiplication and squaring operations. A field multiplication is composed of a "conventional" multiplication of two 192-bit operands, yielding a 384-bit product, followed by a reduction of the product modulo the prime p. In this subsection we focus on multiplication and squaring; the modular reduction operation is subject of the next subsection.

There are two basic algorithms for multi-precision multiplication: one is the *operand-scanning method* (also called row-wise multiplication) and the other is the *product-scanning method* (column-wise multiplication) [18,21]. Both require the same number of single-precision multiplications (i.e. `mul` instructions on an ATmega128), namely 576 in our case of 192-bit operands, but they differ in the number of memory accesses and single-precision additions. We first describe the original operand and product scanning methods, which operate on 8-bit words (i.e. bytes) when implemented on an ATmega processor. Later in this section we

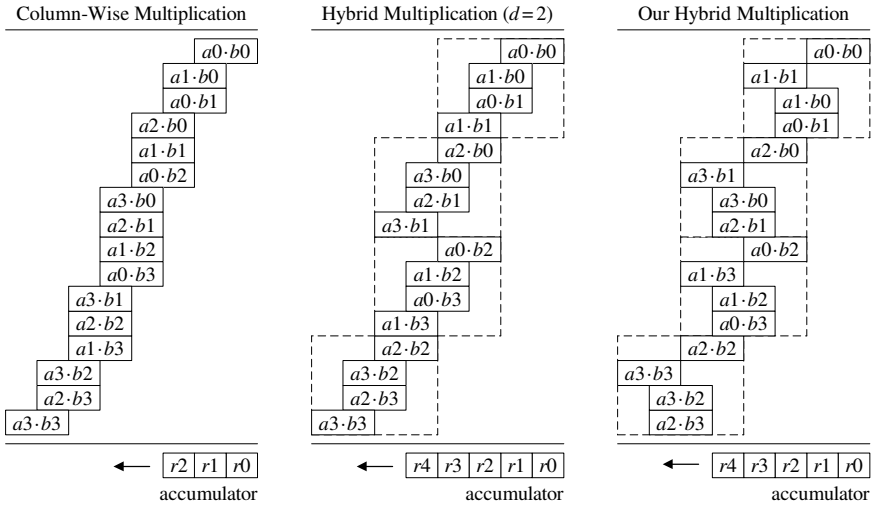

Fig. 1. Comparison between the conventional product-scanning method (left), Gura's hybrid multiplication (middle), and our variant of hybrid multiplication (right).

introduce our optimized version that uses 16-bit words as smallest unit of data it operates on. The operand-scanning method has a nested-loop structure with a relatively simple inner loop. Each iteration executes an operation of the form $a \cdot b + c + d$ with a, b, c, d denoting 8-bit words (i.e. bytes). On an ATmega this operation requires one `mul` instruction to produce the partial product $a \cdot b$, and a total of four `add` (resp. `adc`) instructions to add the two bytes c and d to the 16-bit quantity $a \cdot b$. Furthermore, two load (`ld`) instructions and a store (`st`) are executed in each iteration. On the other hand, the product-scanning method performs a multiply-accumulate operation in its inner loop, i.e. two bytes are multiplied and the 16-bit partial product is added to a cumulative sum held in three registers, as illustrated on the left side of Figure 1. The product-scanning method also executes two `ld` instructions per iteration, but no store [18].

The execution time of the conventional product-scanning method can be significantly improved if the processor features a large number of general-purpose registers, which is the case with the ATmega128 [2]. The *hybrid multiplication method*, introduced by Gura et al in [21], works similar as the product-scanning technique, but operates on words consisting of $d \geq 2$ bytes, which reduces the number of required loop iterations by a factor of d. Figure 1 shows an example for 32-bit operands and $d = 2$. In each iteration of the inner loop a 16-bit word (i.e. two bytes) of operand a and a 16-bit word of operand b are loaded from memory. These two 16-bit words are multiplied using the `mul` instruction, and the product is added to a cumulative sum held in five registers. The rectangles in Figure 1 represent 16-bit products as obtained by the multiplication of two bytes. The four `mul` instructions needed for a multiplication of two 16-bit words are actually executed in row-wise order. Gura et al state in [21] that the hybrid

method employs the product-scanning strategy as the "outer algorithm" and the operand-scanning strategy as the "inner algorithm" (i.e. for the (8×8)-bit `mul` instructions within a $(d \times d)$-byte multiplication). When multiplying two 192-bit operands, the hybrid method with $d = 2$ executes 576 `ld` instructions, which represents a 50% improvement over the standard product-scanning technique [21]. The number of `mul` instructions remains the same.

Our implementation of the hybrid multiplication aims at reducing the number of `add` (resp. `adc`) instructions compared to Gura et al's method. To achieve this, we employ the product-scanning strategy as the "inner algorithm," but schedule the `mul` instructions in a non-conventional order such that the addition to the cumulative sum (including carry propagation) can be performed in "one pass" for several 16-bit partial products. For example, let us have a look at the multiplication of the 16-bit word (a_1, a_0) by the 16-bit word (b_1, b_0), depicted in the top right of Figure 1. We first multiply a_0 by b_0 and add the 16-bit partial product $a_0 \cdot b_0$ to the least significant 16 bits of the cumulative sum held in the register pair r_1, r_0. The second `mul` instruction produces the partial product $a_1 \cdot b_1$, which is added to the content of registers r_4, r_3, r_2. Hence, the addition of the two partial products $a_0 \cdot b_0$, $a_1 \cdot b_1$ to the cumulative sum requires only five `add` (resp. `adc`) instructions altogether. Our method can be easily applied for hybrid multiplication with $d = 4$; in this case 51 `add/adc` instructions are executed per iteration of the inner loop. Unfortunately, when implemented on the ATmega, our method requires to perform a number of `movw` instructions to copy the results of the `mul` instructions to pairs of temporary registers, which is necessary since `mul` overwrites the carry flag. Scott and Szczechowiak describe in [35] a similar hybrid method using so-called "carry-catcher" registers.

The square a^2 of a multiple-precision integer a can be computed significantly faster than the product $a \cdot b$ of two distinct integers. Due to a "symmetry" in the squaring operation, most partial products appear twice. However, they need only be computed once and then left-shifted in order to be doubled. Our optimized squaring routine executes just 300 `mul` instructions for a 192-bit operand.

3.3 Modular Reduction

Each 384-bit product (or square) needs to be reduced modulo $p = 2^{192} - 2^{64} - 1$ to get the final result. This modular reduction operation can be implemented very efficiently since p is a generalized-Mersenne prime. In fact, the reduction requires only three 192-bit additions, followed by a few conditional subtractions of p to get a reduced result that is at most 192 bits long (see [22] for a detailed treatment). We implemented the reduction operation as described in [19].

4 Experimental Results and Discussion

We developed a simple TinyOS program for ECDH key exchange and executed it on a MICAz mote, which allowed us to analyze the running time and energy consumption of the protocol. Each key exchange requires the mote to perform

Table 1. Runtime comparison of different implementations of scalar multiplication.

Implementation	Finite field	Fixed P.	Rand. P.	Notes
Blaß and Zitterbart [4]	GF(2^m), 113 bit	6.74 s	17.28 s	comb, dbl-and-add
Malan et al. [29]	GF(2^m), 163 bit	34.17 s	34.17 s	double-and-add
Yan and Shi [43]	GF(2^m), 163 bit	13.9 s	13.9 s	affine coordinates
Seo et al. [36]	GF(2^m), 163 bit	1.14 s	1.14 s	Koblitz curve
Kargl et al. [23]	GF(2^m), 167 bit	0.763 s	0.763 s	Montgomery ladder
Wang and Li [41]	GF(p), 160 bit	1.24 s	1.35 s	sld. window, NAF
Szczechowiak et al. [38]	GF(p), 160 bit	1.27 s	1.27 s	comb method
Ugus et al. [39]	GF(p), 160 bit	0.57 s	1.03 s	comb, window
Liu and Ning [25]	GF(p), 192 bit	2.99 s	2.99 s	sliding window
Gura et al. [21]	GF(p), 192 bit	1.35 s	1.35 s	NAF
Fürbass et al. [17]	GF(p), 192 bit	0.068 s	0.068 s	hardware impl.
Our implementation	GF(p), 192 bit	0.71 s	1.67 s	comb, window

two point multiplications, one with a fixed point (to generate an ephemeral key pair), and the second with a random point (to obtain the shared secret). The former uses a fixed-base comb method with 14 pre-computed points and can be carried out in $5.20 \cdot 10^6$ clock cycles (0.71 sec), while the latter is implemented using a window method with a window size of 4 (i.e. 14 pre-computed points) and executes in $12.33 \cdot 10^6$ cycles (1.67 sec). Based on the energy characteristics of the MICAz mote [34,13], these timings translate into an energy consumption of 17.04 mJ and 40.08 mJ, respectively. The energy cost of transmitting one protocol message is 0.205 mJ, which means that the total energy consumption of our ECDH key exchange is approximately 57.33 mJ per node. Taking again the energy model from [34] as reference, we can perform 117,750 key exchanges before the battery voltage drops below the value needed by the ATmega128.

Our evaluation shows that the overall energy cost of ECDH key exchange is primarily determined by the computation of the two scalar multiplications on each node; the energy needed for radio communication is almost negligible. We also conducted experiments with point compression [22], a technique that allows to represent a point using the minimum possible number of bits so as to reduce the energy cost of radio communication in ECDH key exchange. However, on the MICAz mote, point compression did not yield any savings in energy.

A comparison with related work (see Table 1) shows that our implementation is significantly faster than most previously-reported 192-bit implementations on 8-bit AVR processors and outperforms even some 160-bit implementations. This performance gain is primarily due to the efficient implementation of the field arithmetic (in particular the field multiplication) and the use of the window and comb methods with a window size of 4 for scalar multiplication. The additional memory demand of these methods is small enough to let our TinyOS program for ECDH key exchange fit into the 4 kB RAM of the MICAz mote. Despite all resource constraints, our software implementation of the comb method is only by a factor of 10 slower than the hardware implementation reported in [17].

Protection Against Side-Channel Attacks. Side-channel attacks belong to the genre of implementation attacks and use information leaking from a device while it executes a cryptographic algorithm (e.g. power consumption, execution time) to reveal the secret key [22]. Fortunately, ECDH key exchange is not vulnerable to DPA and timing attacks as the scalar multiplications are performed with new random numbers in each run of the protocol. However, an SPA attack on the scalar multiplication is possible, and if successful, provides the attacker with the random number k that is part of the ephemeral key pair generated in each run of the protocol (see Section 2.1), which enables him to eavesdrop on the communication between the nodes.

In order to foil SPA attacks, the scalar multiplication should be implemented in such a way that always the same sequence of operations (i.e. point additions and doublings) is executed, independent of the scalar. Of course, this requires an SPA-resistant implementation of the field arithmetic too. For example, small irregularities in the modular addition or modular reduction (e.g. conditional subtractions of the prime p) typically lead to differences in execution time and power consumption, which can be exploited to mount an SPA attack [22]. It is particularly important to prevent conditional subtractions in the fast reduction operation; we achieved this by following the approach from [19].

As described in Subsection 2.2, we use a window method with a window size of 4 to implement the scalar multiplication by an arbitrary base point P, and a fixed-base comb method if P is known a-priori. Both methods can be made SPA-resistant by converting the scalar k into a radix-2^4 representation with a digit-set that does not contain 0. Such conversions are easy to implement and have only little impact on performance (about 5% in our implementation).

5 Conclusions

We presented an optimized implementation of ECDH key exchange for MICAz motes. Our implementation utilizes a NIST-recommended elliptic curve over a 192-bit prime field as underlying algebraic structure and executes a full scalar multiplication in 0.71 sec ($5.20 \cdot 10^6$ cycles) when the base point is fixed and known a priori. A scalar multiplication by an arbitrary base point takes 1.67 sec ($12.33 \cdot 10^6$ cycles). The total amount of energy required to perform an ECDH key exchange is approximately 57 mJ per node, which means that each node can carry out over 117,000 key exchanges before running out of battery.

Our ECDH key exchange is significantly faster and more energy-efficient than previously-reported 192-bit implementations on comparable 8-bit platforms. The higher performance is mainly due to the use of the window and comb methods for scalar multiplication instead of the simple double-and-add technique. The additional memory demand when using a window method with a window size of 4 is relatively small (1080 bytes) and fits easily into the motes' 4 kB RAM. In addition, the window method can be made SPA-resistant without much loss in performance. Putting it all together, our results confirm that high performance *and* side-channel resistivity can be achieved on resource-constrained motes.

Acknowledgements. The research described in this paper has been supported by the EPSRC under grant EP/E001556/1, the Austrian ministry BM:VIT in the FIT-IT program line "Trust in IT Systems" under grant 816151 (project POWER-TRUST), and the European Commission under grant FP6-IST-033563 (project SMEPP) and, in part, through the ICT Programme under contract ICT-2007-216676 ECRYPT II. The information in this document reflects only the authors' views, is provided as is, and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. Atmel Corporation. 8-bit ARV® Instruction Set. User Guide, available for download at `http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf`, July 2008.
2. Atmel Corporation. 8-bit ARV® Microcontroller with 128K Bytes In-System Programmable Flash: ATmega128, ATmega128L. Datasheet, available for download at `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`, June 2008.
3. A. Becher, Z. Benenson, and M. Dornseif. Tampering with motes: Real-world physical attacks on wireless sensor networks. In *Security in Pervasive Computing — SPC 2006*, vol. 3984 of *Lecture Notes in Computer Science*, pp. 104–118. Springer Verlag, 2006.
4. E.-O. Blaß and M. Zitterbart. Efficient implementation of elliptic curve cryptography for wireless sensor networks. Technical Report TM-2005-1, Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Mar. 2005. Available for download at `http://doc.tm.uka.de/2005/tm-2005-1.pdf`.
5. C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly-secure key distribution for dynamic conferences. In *Advances in Cryptology — CRYPTO '92*, vol. 740 of *Lecture Notes in Computer Science*, pp. 471–486. Springer Verlag, 1993.
6. H. Brody. 10 emerging technologies that will change the world. *Technology Review*, 106(1):33–49, Feb. 2003.
7. H. Chan and A. Perrig. Security and privacy in sensor networks. *Computer*, 36(10):103–105, Oct. 2003.
8. H. Chan and A. Perrig. PIKE: Peer intermediaries for key establishment in sensor networks. In *Proceedings of the 24th IEEE International Conference on Computer Communications (INFOCOM 2005)*, vol. 1, pp. 524–535. IEEE, 2005.
9. H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *Proceedings of the 24th IEEE Symposium on Security and Privacy (S&P 2003)*, pp. 197–213. IEEE Computer Society Press, 2003.
10. J. P. Conti. The Internet of things. *IET Communications Engineer*, 4(6):20–25, Dec./Jan. 2006/2007.
11. Crossbow Technology, Inc. MICAz Wireless Measurement System. Data sheet, available for download at `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`, 2006.
12. S. K. Das, A. Agah, and K. Basu. Security in wireless mobile and sensor networks. In *Wireless Communications Systems and Networks*, chapter 18, pp. 531–557. Springer Verlag, 2004.

13. G. de Meulenaer, F. Gosset, F.-X. Standaert, and O. Pereira. On the energy cost of communication and cryptography in wireless sensor networks. In *Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WIMOB 2008)*, pp. 580–585. IEEE Computer Society Press, 2008.
14. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov. 1976.
15. W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pp. 62–72. ACM Press, 2003.
16. L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, pp. 41–47. ACM Press, 2002.
17. F. Fürbass and J. Wolkerstorfer. ECC processor with low die size for RFID applications. In *Proceedings of the 40th IEEE International Symposium on Circuits and Systems (ISCAS 2007)*, pp. 1835–1838. IEEE, 2007.
18. J. Großschädl, R. M. Avanzi, E. Savaş, and S. Tillich. Energy-efficient software implementation of long integer modular arithmetic. In *Cryptographic Hardware and Embedded Systems — CHES 2005*, vol. 3659 of *Lecture Notes in Computer Science*, pp. 75–90. Springer Verlag, 2005.
19. J. Großschädl and E. Savaş. Instruction set extensions for fast arithmetic in finite fields GF(p) and GF(2^m). In *Cryptographic Hardware and Embedded Systems — CHES 2004*, vol. 3156 of *Lecture Notes in Computer Science*, pp. 133–147. Springer Verlag, 2004.
20. J. Großschädl, A. Szekely, and S. Tillich. The energy cost of cryptographic key establishment in wireless sensor networks. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS 2007)*, pp. 380–382. ACM Press, 2007.
21. N. Gura, A. Patel, A. S. Wander, H. Eberle, and S. Chang Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, vol. 3156 of *Lecture Notes in Computer Science*, pp. 119–132. Springer Verlag, 2004.
22. D. R. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Verlag, 2004.
23. A. Kargl, S. Pyka, and H. Seuschek. Fast arithmetic on ATmega128 for elliptic curve cryptography. Cryptology ePrint Archive, Report 2008/442, 2008.
24. J. T. Kohl and B. C. Neuman. The Kerberos Network Authentication Service (Version 5). Internet Engineering Task Force, Network Working Group, RFC 1510, Sept. 1993.
25. A. Liu and P. Ning. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008)*, pp. 245–256. IEEE Computer Society Press, 2008.
26. D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pp. 52–61. ACM Press, 2003.
27. D. Liu and P. Ning. *Security for Wireless Sensor Networks*, vol. 28 of *Advances in Information Security*. Springer Verlag, 2006.
28. J. Lopez and J. Zhou. *Wireless Sensor Network Security*, vol. 1 of *Cryptology and Information Security Series*. IOS Press, 2008.

29. D. J. Malan, M. Welsh, and M. D. Smith. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. In *Proceedings of the 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004)*, pp. 71–80. IEEE, 2004.
30. National Institute of Standards and Technology (NIST). Recommended Elliptic Curves for Federal Government Use. White paper, available for download at `http://csrc.nist.gov/encryption/dss/ecdsa/NISTReCur.pdf`, July 1999.
31. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, Dec. 1978.
32. ON World, Inc. WSN for smart homes. Market Dynamics Report, Feb. 2008.
33. A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM 2001)*, pp. 189–199. ACM Press, 2001.
34. K. Piotrowski, P. Langendörfer, and S. Peter. How public key cryptography influences wireless sensor node lifetime. In *Proceedings of the 4th ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2006)*, pp. 169–176. ACM Press, 2006.
35. M. Scott and P. Szczechowiak. Optimizing multiprecision multiplication for public key cryptography. Cryptology ePrint Archive, Report 2007/299, 2007.
36. S. C. Seo, D.-G. Han, H. C. Kim, and S. Hong. TinyECCK: Efficient elliptic curve cryptography implementation over GF(2^m) on 8-bit Micaz mote. *IEICE Transactions on Information and Systems*, E91-D(5):1338–1347, May 2008.
37. A. Swami, Q. Zhao, Y.-W. Hong, and L. Tong. *Wireless Sensor Networks: Signal Processing and Communications Perspectives*. John Wiley and Sons Ltd, 2007.
38. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In *Wireless Sensor Networks — EWSN 2008*, vol. 4913 of *Lecture Notes in Computer Science*, pp. 305–320. Springer Verlag, 2008.
39. O. Ugus, D. Westhoff, R. Laue, A. Shoufan, and S. A. Huss. Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor networks. In *Proceedings of the 2nd Workshop on Embedded Systems Security (WESS 2007)*, pp. 11–16, 2007.
40. L. Uhsadel, A. Poschmann, and C. Paar. Enabling full-size public-key algorithms on 8-bit sensor nodes. In *Security and Privacy in Ad-hoc and Sensor Networks — SASN 2007*, vol. 4572 of *Lecture Notes in Computer Science*, pp. 73–86. Springer Verlag, 2007.
41. H. Wang and Q. Li. Efficient implementation of public key cryptosystems on mote sensors. In *Information and Communications Security — ICICS 2006*, vol. 4307 of *Lecture Notes in Computer Science*, pp. 519–528. Springer Verlag, 2006.
42. Y. Xiao, V. K. Rayi, B. Sun, X. Du, F. Hu, and M. Galloway. A survey of key management schemes in wireless sensor networks. *Computer Communications*, 30(11–12):2314–2341, Sept. 2007.
43. H. Yan and Z. J. Shi. Studying software implementations of elliptic curve cryptography. In *Proceedings of the 3rd International Conference on Information Technology: New Generations (ITNG 2006)*, pp. 78–83. IEEE Computer Society Press, 2006.
44. S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: A probabilistic approach. In *Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP 2003)*, pp. 326–335. IEEE Computer Society Press, 2003.