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

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 introduce an improved variant of the product scanning technique for multiple-precision multiplication [22] that reduces the number of load operations (i.e. `ld` instructions on an ATmega128L processor [2]). Our variant is similar (but not identical) to the hybrid multiplication techniques described in [21], [35], and [40]. Second, our implementation uses advanced window methods for scalar multiplication to reduce execution time at the expense of a slight increase in memory requirements. However, we show that despite the additional memory demand, the window methods are perfectly feasible for MICAz motes. Third, we aimed to secure our ECDH implementation against side-channel attacks.

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 with x and y referred to as coordinates. The set of points together with a special point (the so-called point at infinity) forms an Abelian group if the addition is suitably defined.

Two operations are defined over an elliptic curve: addition and doubling of points. Given an additional point $\mathcal{O}$, the so-called "point-at-infinity", the points on an elliptic curve defined over a finite field form an additive Abelian group. Figures 1 and 2 give a graphical representation of these operations. Note that each group operations requires several operations in the underlying field, so the efficiency and performance of operations over the curve heavily depends on fast implementations of the field operations.

The basic building block of public key cryptographic systems is the (assumed) intractability of a hard mathematical problem, like for example the Discrete Logarithm Problem (DLP), that is, given $x = a^y \bmod n$ and knowing x, a and n, it is impossible for an attacker to determine y. The widely used RSA system uses this approach, where a keylength of 1,024 bit is assumed to provide a decent level of security.

The equivalent to this problem on elliptic curves is the Elliptic Curve Discrete Logarithm Problem (ECDLP), that is, given $Q = k * P$ with Q and P points on the curve and k an arbitrary integer, it is assumed to be impossible to determine k from the knowledge of Q and P. Since until today there is no efficient algorithm available to solve the ECDLP, an ECC cryptographic system with 160 bit is regarded to be at least as safe as an RSA system with 1,024 bit [22]. This makes the use of ECC especially attractive in constrained environments like mobile phones, PDAs, smart cards, embedded devices, and sensor motes.

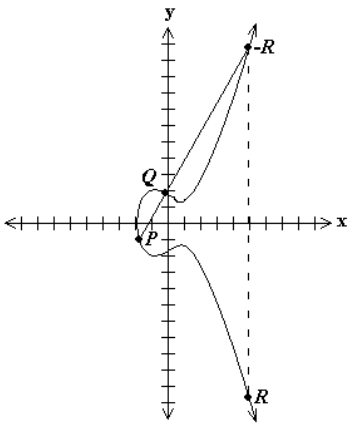

Fig. 1. $R = P + Q$

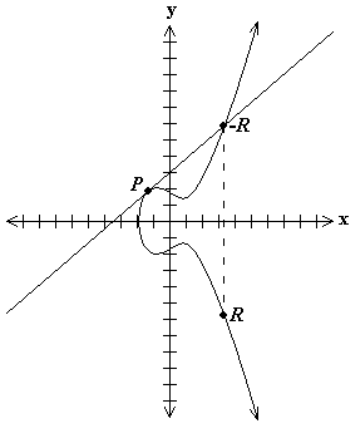

Fig. 2. $R = 2P$

An elliptic curve domain is completely described by the tuple (q, a, b, P, n, h), where P is a generator of order n for a cyclic subgroup on the curve with shape parameters a and b. q is the order of the underlying field and h the co-factor, that is, the number of points on the curve divided by n. For practical implementations, h should be as close to one as possible. Our work focuses on the curve over the NIST prime field P_{192}.

2.1 Elliptic Curve Diffie-Hellman Protocol

The elliptic curve equivalent of the Diffie-Hellmann key exchange relies on the intractability of the ECDLP as described in Section 2 and is used to establish a shared secret between two entities using an insecure channel.

Figure 3 shows a timing diagram of a single ECDH protocol run, where Alice and Bob establish a shared secret. An attacker can intercept Q_A and Q_B, but it is assumed that he is unable to calculate k and l from Q_A and Q_B, respectively.

So, a total of four point multiplications is necessary to create the secret key, two multiplications for each node. The first two could be performed simultaneously by Alice and Bob, the last two have to be performed in sequence. It is also possible to precalculate a new Q_A and Q_B when the nodes are idling to speed up subsequent protocol runs.

A straightforward implementation of ECDH key exchange is vulnerable to the man-in-the-middle attack. 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

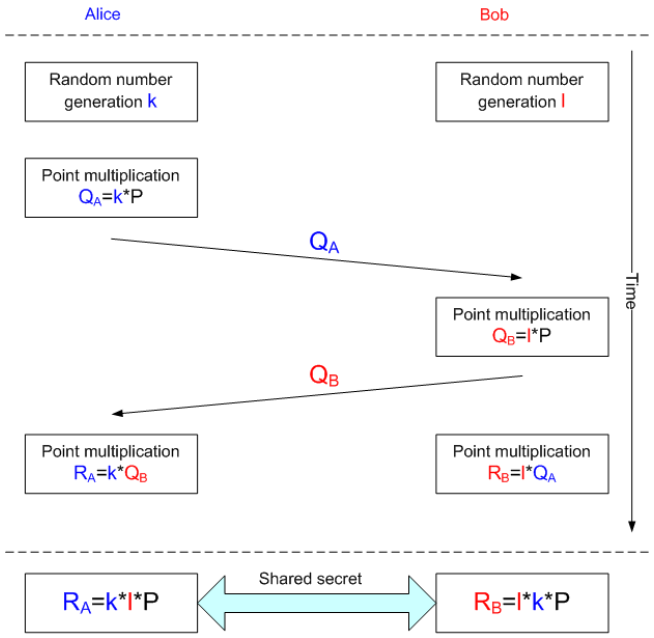

Fig. 3. Timing Diagram of the ECDH Key Establishment

ECDH protocol incorporating authentication, such as Signed ECDH or ECMQV, has been studied in [13] and [20].

3 Prime-Field Arithmetic

For our implementation we use the NIST prime field F_{192} [30], that is $p_{192} = 2^{192} - 2^{64} - 1$. Since we are working on an 8-bit architecture, we would need 24 registers to hold an entire element of this finite field. All field operations are implemented in assembly and optimized for low cycle counts, especially the modular multiplication and squaring, which have the biggest influence on the running time of the group operations on the curve (a single point addition requires eight field multiplications and three field squarings, a point doubling four field multiplications and squarings, respectively).

3.1 Modular Multiplication

The performance of the scalar point multiplication depends a lot on the performance of the field multiplication. In our case we have to multiply two 192-bit operands, giving a 384-bit result. The microcontroller on our sensor motes offers an instruction which allows to multiply two 8-bit numbers, yielding a 16-bit result [1]. We use this instruction repeatedly to compute all the partial products necessary to obtain the final 384-bit multiplication result. In case of 192-bit

operands, 576 16-bit partial products have to be calculated. These partial products then have to be accumulated to give the final result.

To perform multiplication efficiently we decided to implement an algorithm which is similar to the well-known product scanning method [22]. In each iteration of the inner loop, the product scanning method fetches *two* 8-bit operands from memory and calculates *one* 16-bit partial product [18]. In contrast to this, our algorithm fetches *four* 8-bit operands and calculates *four* partial products. Hence the conventional product scanning method needs two LD (load) instructions per computed partial product, while our algorithm needs only one LD (load) instruction per calculated partial product. That means that the number of memory accesses can be reduced by a factor of two. The drawback of this solution is that we need to use a larger accumulator which consists of five registers, while an accumulator using three registers is sufficient in case of standard product scanning. The usage of this larger accumulator makes it also necessary to apply more ADC (add with carry) instructions to propagate carries.

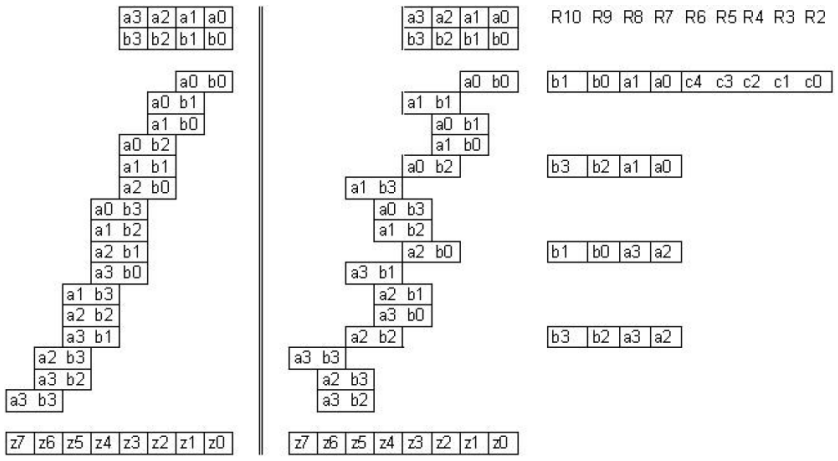

Fig. 4. Comparison between the standard product scanning method (on the left) and our approach (on the right).

Despite to this drawback the advantages of our solution prevail. We save 1.152 cycles for LD (load) instructions and loose only 144 cycles for additional ADC (add with carry) instructions. Figure 4 compares the implemented algorithm to the usual product scanning algorithm. The illustration displays both algorithms for 32-bit operands, giving a 64-bit result. The operands a and b are composed of 8-bit parts a_i and b_i, respectively. $a_i b_j$ represent partial products which are computed and accumulated one after another. The result z is composed of 8-bit z_is. We can see that both algorithms calculate the same partial products, but the sequence of computation is different.

3.2 Modular Squaring

Together with field multiplication, field squaring has a significant impact on the performance of the scalar point multiplication. The result of squaring a 192-bit number is a 384-bit number. The squaring algorithm is similar to the multiplication algorithm, but exploits a property to achieve a performance gain; the implementation of squaring uses the 8-bit multiplication instruction of the microcontroller 300 times, while the multiplication uses the same instruction 576 times to calculate the partial products.

If we want to multiply two distinct sums consisting of two summands, we have to compute four partial products:

$(a_0 + a_1) * (b_0 + b_1) = a_0 * b_0 + a_0 * b_1 + a_1 * b_0 + a_1 * b_1$

If we want to square a sum consisting of two summands we need to compute only three partial products. The partial product $a_0 * a_1$ can be used twice (equivalent to a cheap left-shift operation):

$(a_0 + a_1)^2 = a_0 * a_0 + 2 * a_0 * a_1 + a_1 * a_1$

Figure 5 illustrates how the squaring algorithm works for a 32-bit operand. The result is 64 bit long. The operand a consists of 4 8-bit parts a_i. $a_i a_j$ represent partial products. At first we compute those partial products which are used twice and accumulate them. When the intermediate result is computed, we shift the whole intermediate result left, which is much faster than an additional multiplication operation. When shifting is accomplished, we compute those partial products which are only used once and accumulate them. Finally we end up with a result in memory which is 64 bit long, depicted by the z_is.

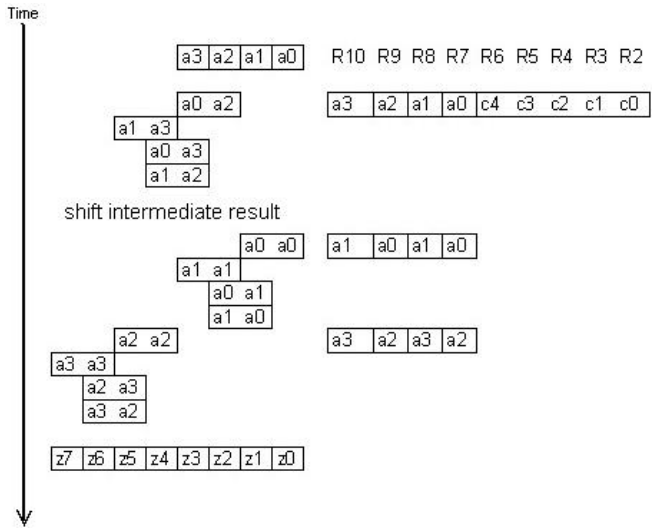

Fig. 5. Squaring exploits the fact that some partial products can be used twice. Those partial products are computed and accumulated initially. Than the intermediate result is left-shifted. Finally the remaining partial products are computed and accumulated.

3.3 Fast Reduction

After each multiplication or squaring, a modular reduction has to be performed to achieve a valid field element again. In our case, we have to reduce modulo the prime number $p_{192} = 2^{192} - 2^{64} - 1$. There exist very efficient algorithms for reductions for moduli with this special structure, called Fast Reduction. These algorithms are not generic like for example Montgomery Reduction and each one only works for one certain modulus. It can be executed much faster than generic implementations.

Our modulus has the form: $2^{192} - 2^{64} - 1 \equiv 0 \mod p_{192}$. This equation can be transformed to $2^{192} \equiv 2^{64} + 1 \mod p_{192}$.

The number Z to reduce is 384 bit long. It can be represented by two 192-bit parts, a most significant part Z_H and a least significant part Z_L:

$Z = Z_H * 2^{192} + Z_L$. The result of our algorithm is the 192-bit number R $R = Z_H * 2^{192} + Z_L \mod p_{192}$. This equation can be rewritten by making a simple substitution and we obtain the intermediate result. This intermediate result can be represented by a most significant part $Z_H^{'}$ and a least significant part $Z_L^{'}$.

$R = Z_H * (2^{64} + 1) + Z_L \mod p_{192}$. $R = Z_H^{'} * 2^{192} + Z_L^{'} \mod p_{192}$.

We can substitute again and obtain our final result. We see that we can obtain the result of our modular reduction by simply performing three 192 bit additions.

$R = Z_H^{'} * (2^{64} + 1) + Z_L^{'} \mod p_{192}$. $R = Z_H^{'} * 2^{64} + Z_H^{'} * 1 + Z_L^{'} \mod p_{192}$.

Figure 6 illustrates the implemented algorithm. Z is the 384-bit number to be reduced, consisting of 32-bit Z_is. The reduced result R can be obtained by an iterative substitution, leading to three 192-bit additions. In rare cases, one or two bits of R_6 are set after the three additions, requiring a final subtraction of the modulus [19].

4 Group Arithmetic

The basic and most important operation for cryptographic algorithms over a group of points on an elliptic curve is the scalar multiplication, that is the repeated application of point doubling and point addition to achieve a point $Q = k*P$. The straightforward implementation of this is the elliptic curve equivalent of the square-and-multiply approach taken in RSA, namely the double-and-add algorithm; for each bit in the scalar k, the current result is doubled, for each set bit in k, P is added to the result. For an n-bit scalar this results in average in n point doublings and $n/2$ point additions.

A simple modification of this technique is to alter the structure of k by using a signed representation instead of the ordinary binary one, the Non Adjacent Form (NAF) ([22]). Since subtraction of P is equivalent to adding $-P$, and the negation of a point is almost free, the NAF tries to minimize the number of non-zero bits in k. In average, this approach brings down the number of required point additions (either of P or $-P$) to $n/3$. The number of point doublings remains

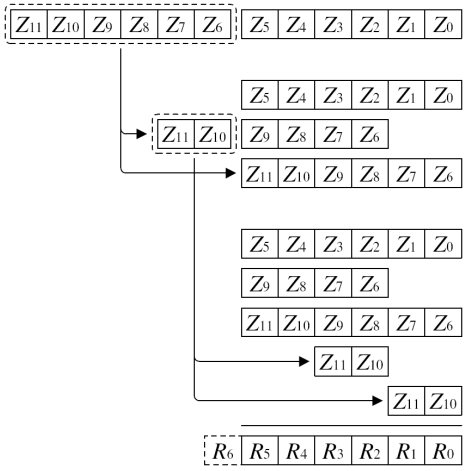

Fig. 6. Structure of the Fast Reduction for a Modulus of the Form $2^{192} - 2^{64} - 1$.

unchanged. This technique can also be extended to windowed approaches, by not only processing one bit of the scalar at a time, but w, for the cost of additional storage for the precomputed $2^w - 2$ points. We chose a window width of four, so precomputation required 1.080 bytes of storage.

If the point P is known beforehand (as it is the case in the first two multiplications when performing ECDH, see Figure 3) this precomputation can be even done offline to further speed up the entire scalar operation. We use the windowed-NAF method for the last two multiplications and the windowed-NAF method with precomputed multiples of P for the first two multiplications during an ECDH protocol run.

5 Implementation and Results

For the actual implementation we used MICAz motes, equipped with an 8-bit ATmega128L microcontroller running at 7.3728 MHz. We had 128kb of in-system reprogrammable flash memory at our disposal, in which we had to fit TinyOS and our extensions to perform the ECDH key exchange protocol. Table 1 shows the runtime of a single scalar multiplication we could achieve with our implementation, compared to some other recent and well known solutions. An entire ECDH run consists, as described in Section 4, of a fixed point and a random point multiplication for each node.

In addition to the running time, we try to give a simple lifetime calculation, based on the estimations in [34]. Assuming the energy required for a random point multiplication to be 40.08 mJ and that for a fixed point multiplication to be 17,04 mJ (by considering the individual running times), and since we know the total energy required to transmit one protocol message to be 0,205 mJ, our

Table 1. Runtime Comparison of the Point Multiplication

	Fixed point	Random point	Conditions
Malan et al. [29]	34.17s	34.17s	$GF(2^m)$, 163-bit
Yan and Shi [43]	13.9s	13.9s	$GF(2^m)$, 163-bit
Blaß and Zitterbart [4]	6.74s	17.28s	$GF(2^m)$, 113-bit
Seo et al. [36]	1.14	1.14	$GF(2^m)$, 163 bit, Koblitz curve
Kargl et al. [23]	0.69	1.10	$GF(2^m)$, 167-bit
Wang and Li [41]	1.24s	1.35s	$GF(p)$, 160-bit
Gura et al. [21]	1.35s	1.35s	$GF(p)$, 192-bit
Szczechowiak et al. [38]	1.27s	1.27s	$GF(p)$, 160-bit
Liu and Ning [25]	2.99s	2.99s	$GF(p)$, 192-bit, hashing
Uhsadel et al. [40]	0.76s	0.76s	$GF(p)$, 160-bit, estimated
Ugus et al. [39]	0.57	1.03	$GF(p)$, 160-bit
Fürbass et al. [17]	0.068	0.068	$GF(p)$, 192-bit, hardware impl.
This Implementation	0.71s	1.67s	$GF(p)$, 192-bit

implementation allows 117,750 key exchanges before the battery voltage drops below the value needed by our microcontroller.

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 in order to reduce the energy cost of radio communication in ECDH key exchange. However, on the MICAz mote, point compression is not attractive due to the costly computations needed to recover the original representation of the point.

A comparison with related work (see Table 1) shows that our implementation is significantly faster than previously-reported 192-bit implementations and outperforms even some 160-bit implementations. This performance gain is primarily due to the use of window methods for scalar multiplication. The additional memory requirements of the windows methods are moderate enough to let the resulting TinyOS program fit into the 4 kB RAM of the MICAz mote.

5.1 Resistance Against Side-Channel Attacks

Fortunately, ECDH is not vulnerable to timing and DPA attacks as the scalar multiplications are performed with a new random numbers in each run of the protocol. However, an SPA attack on the scalar multiplications is possible, and if successful, provides the attacker with the random number(s), which eventually 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 as well. For example, irregularities in the implementation of modular addition and modular multiplication

(e.g. conditional subtractions of the prime P) lead to differences in execution time and power consumption, which can be exploited to mount an SPA attack [22]. Of particular importance is to prevent conditional subtractions of P in the fast reduction; we achieved this by following the approach from [19].

As mentioned in Section 4, we use a window method with a window size of 4 to implement the scalar multiplication. This method scans four bits of the scalar at once, and adds the corresponding multiple from the set of pre-computet points $\{P, 2P, 3P, \ldots, 15P\}$. However, this requires special consideration of the case when all four bits are 0; in this case no point addition is performed. In order to overcome this irregularity, one can convert the scalar into a radix-2^4 representation with a digit-set that does not contain 0, which makes it possible to eliminate this irregularity. Such conversions are described in [22] and easy to implement. The performance penalty resulting from making the key exchange SPA-resistant is relatively small (about 5% in our implementation).

6 Conclusion

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 based 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 platforms. This improved performance is mainly due to the use of advanced window methods for scalar multiplication instead of the basic double-and-add method. The additional memory demand when using a window method with a window size of 4 is relatively small (just 1080 bytes) and fits easily into the motes' 4 kB RAM. Furthermore, window methods can be made SPA resistant without significant performance degradation. In summary, our results show that high performance *and* side-channel resistivity can be achieved on resource-constraint motes.

Acknowledgements

The research described in this paper has been supported by the 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 J. A. Clark, R. F. Paige, F. Polack, and P. J. Brooke, editors, *Security in Pervasive Computing — SPC 2006*, volume 3984 of *Lecture Notes in Computer Science*, pages 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 E. F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 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)*, volume 1, pages 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)*, pages 197–213. IEEE Computer Society Press, 2003.
10. J. P. Conti. The Internet of things. *IET Communications Engineer*, 4(6):20–25, December/January 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`, Jan. 2006.
12. S. K. Das, A. Agah, and K. Basu. Security in wireless mobile and sensor networks. In M. Guizani, editor, *Wireless Communications Systems and Networks*, chapter 18, pages 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)*, pages 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 S. Jajodia, V. Atluri, and T. Jaeger, editors, *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pages 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)*, pages 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)*, pages 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 J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 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 M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 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 R. H. Deng and P. Samarati, editors, *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS 2007)*, pages 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 M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 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. Avialable for download at `http://eprint.iacr.org`.
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)*, pages 245–256. IEEE Computer Society Press, 2008.
26. D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pages 52–61. ACM Press, 2003.
27. D. Liu and P. Ning. *Security for Wireless Sensor Networks*, volume 28 of *Advances in Information Security*. Springer Verlag, 2006.
28. J. Lopez and J. Zhou. *Wireless Sensor Network Security*, volume 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)*, pages 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)*, pages 189–199. ACM Press, 2001.
34. K. Piotrowski, P. Langendörfer, and S. Peter. How public key cryptography influences wireless sensor node lifetime. In S. Zhu and D. Liu, editors, *Proceedings of the 4th ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2006)*, pages 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. Avialable for download at `http://eprint.iacr.org`.
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 R. Verdone, editor, *Wireless Sensor Networks — EWSN 2008*, volume 4913 of *Lecture Notes in Computer Science*, pages 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 T. Wolf and S. Parameswaran, editors, *Proceedings of the 2nd Workshop on Embedded Systems Security (WESS 2007)*, pages 11–16, 2007. Available for download at `http://arxiv.org/abs/0903.3900`.
40. L. Uhsadel, A. Poschmann, and C. Paar. Enabling full-size public-key algorithms on 8-bit sensor nodes. In F. Stajano, C. Meadows, S. Capkun, and T. Moore, editors, *Security and Privacy in Ad-hoc and Sensor Networks — SASN 2007*, volume 4572 of *Lecture Notes in Computer Science*, pages 73–86. Springer Verlag, 2007.
41. H. Wang and Q. Li. Efficient implementation of public key cryptosystems on mote sensors. In P. Ning, S. Qing, and N. Li, editors, *Information and Communications Security — ICICS 2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 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)*, pages 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)*, pages 326–335. IEEE Computer Society Press, 2003.