On the Application of Pairing Based Cryptography to Wireless Sensor Networks

Piotr Szczechowiak
School of Electronic Engineering
Dublin City University, Ireland
piotr@eeng.dcu.ie

Anton Kargl
Siemens AG
Corporate Technology
München, Germany
anton.kargl@siemens.com

Michael Scott
School of Computing
Dublin City University, Ireland
mike@computing.dcu.ie

Martin Collier
School of Electronic Engineering
Dublin City University, Ireland
collierm@eeng.dcu.ie

ABSTRACT

Recent research results have shown that Elliptic Curve Cryptography (ECC) is feasible on resource constrained sensor nodes. In this work we demonstrate that the related but more complex primitives of Pairing Based Cryptography (PBC) are also well suited for sensor devices.

We present the first in-depth study on the application and implementation of PBC to Wireless Sensor Networks (WSNs). Our implementations are all the fastest yet reported, and have been implemented across a range of WSN processors. On a system level we investigate the application of a simple non-interactive key exchange scheme that is particularly suitable for many WSN scenarios. We also present a novel variant of the key exchange protocol which can be useful in even more demanding applications, and which partially solves the problem of node compromise attacks.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Algorithms, Performance, Security

Keywords

Wireless Sensor Networks, Security protocols, Pairing-based Cryptography, Implementation

1. INTRODUCTION

Wireless sensor networks (WSNs) are a challenging environment in which to enforce acceptable levels of security. And once they are deployed in large numbers and in mission-critical systems, it will only be a matter of time before they are "hacked" for fun, or suborned for profit, by malicious third-parties. As currently envisaged, WSN nodes will be low powered, and will have a limited battery supported lifetime. This implies that only simple and fast methods of cryptography can be used. Since wireless transmission is a major cause of power drainage, unnecessary information exchange should be avoided at all cost. Therefore the security solution should not introduce a large bandwidth overhead.

Another major problem is that WSN nodes will not have secure storage for cryptographic keys, which means that an active attacker can capture one or more nodes, and subvert them, with potentially catastrophic consequences. In an extreme case the entire network can be captured, and all cryptographic keys can be discovered. Therefore the best that can be hoped for is that the network will be immune to passive attack, and that it can survive in some sense the loss and capture of a certain percentage of its nodes.

An advantage from a security point of view is that in the context of a WSN, there is a clearly identified single "trusted authority", and that is the original deployer of the network. The clearly identified existance of such an entity, which is not always the case in cryptography, makes it easier to develop a solution.

One simple approach would be for the deployer to issue just one shared bootstrap crypto key to all nodes. From this initial state the WSN could self-organise into groups which, under cover of the shared key, might then go on to negotiate new group keys for use in subsequent communication. This solution scales nicely to large networks, and facilitates the addition of new nodes to the network. However it does not provide any kind of node-to-node authentication. Also it fails catastrophically to an active attacker, who captures just one node during the early deployment stage, and with it the shared master key.

An alternative approach is to issue each node with a list of $n-1$ keys, one for each of the other $n-1$ nodes in an n-node network. This does provide authentication, but it requires a lot of memory, and it is not at all trivial to add new nodes to an existing network without recalling existing nodes for a refit.

One approach to the problem of low-powered processors might be to initially restrict the set of cryptographic building blocks to the simplest cryptographic primitives (block ciphers and hash functions), and then try to build a solution from these. A more ambitious strategy is to imagine a close-to-ideal solution, and then somehow try to make it practical.

WiSec'09, March 16–18, 2009, Zurich, Switzerland.

Consider for instance a single secret sN issued to each node, where s is the network deployer's master key, and $N \leftarrow H_1(ID_N)$ is derived via a public hash function H_1 from the node's identity ID_N. We assume that given sN it is not feasible to determine s. Consider also a function $e(A, B)$ which has the property of "bilinearity", that is $e(sA, B) = e(A, sB)$, and which is symmetric in that $e(A, B) = e(B, A)$.

Now any pair of nodes ID_A and ID_B can calculate a common key as $H_3(e(sA, B))$ (as calculated by ID_A) and $H_3(sB, A))$ (as calculated by ID_B), where H_3 is a Key Derivation Function (KDF) derived from a standard one-way hash function. This calculation requires no communication (although the identity of the other node needs to be known – but this is already a requirement for communication between them).

In many ways this identity-based scheme is a perfect solution – it scales well, new nodes can be added at any time, the extra bandwidth overhead associated with the cryptography is zero, and subverting one node does not reveal anything about communication between other pairs of nodes (and indeed even subverting all of the nodes does not reveal s). But initially we have to check if such a system is realisable in the WSN environment and if it can be efficiently implemented on sensor nodes. Therefore it is vital to demonstrate that implementation at acceptable speeds is indeed possible.

Before proceeding, a fundamental question needs to be asked - what level of security is adequate? Clearly there is an intuitive correlation between the level of security achieved, and the processing power required. By its nature and in particular its battery-limited lifetime, it is in our view sufficient that the level of security of a WSN is pitched just beyond that which has been demonstrated to be insecure. Cryptanalysts work hard to set new records, and as they push forward the boundaries of what can be "broken", WSN users can respond by stepping up their own levels of security. While it is true that this may not offer security against the resources of secret and well resourced government organisations, we feel this is unlikely to be a realistic threat to legitimate users.

2. INTRODUCTION TO PAIRINGS

In this section we give some necessary background to the mathematics of elliptic curves and pairing-based cryptography. However the reader more interested in the applications of this mathematics might want to skip ahead to section 3.

Our security solution can be realised using the arithmetic of cryptographic pairings on elliptic curves. In our scheme the sensor node identities are hashed and mapped to points (N) on an elliptic curve. These points are of large prime order r, and in this setting the calculation of s given sN is an instance of the Elliptic Curve Discrete Logarithm Problem (ECDLP), which is known to be hard, and which is indeed the basis of the well-established science of elliptic curve cryptography [29]. An elliptic curve is defined over a field $\mathbb{F}_q$, and written as $E(\mathbb{F}_q)$, where q is a prime power. The number of points on an elliptic curve is equal to $q+1-t$, where t (the trace of the Frobenius) is relatively small. These points form an Abelian group.

To realise our identity-based scheme we need to find a suitable function $e(A, B)$ that operates on two elliptic curve points and has all the properties mentioned earlier. Such a function is called a "pairing". However it is not so easy to find a useful and practical pairing. Only on certain special elliptic curves do useful pairings exist, for example the Tate pairing [3], [11]. Such curves are called "pairing-friendly" elliptic curves and the simplest such curves are the supersingular curves.

2.1 Supersingular Elliptic Curves

If $q = 2^m$ then on a supersingular elliptic curve $E(\mathbb{F}_q)$ the Tate pairing $e(A, B)$ evaluates as an element in $\mathbb{F}_{q^k}$, where in this case k, known as the "embedding degree", can be equal to 4. The parameters A and B are points of order r on the curve, where for the supersingular curve r is a large prime divisor of $2^m \pm 2^{(m+1)/2} + 1$, the number of points on the curve [29]. On these curves the ECDLP problem on $E(\mathbb{F}_q)$ can be converted to a DLP problem over $\mathbb{F}_{q^k}$, where fast index calculus algorithms can be used in the attempt to solve the DLP. (Indeed this was one of the first applications of pairings, and the reason why supersingular curves were once so strongly deprecated [29]). For pairings to be secure clearly both problems must be hard for acceptable levels of security.

Other supersingular curves are also possible, over $\mathbb{F}_{3^m}$ where the embedding degree can be 6, or over the prime field $\mathbb{F}_p$ where k is 2. However to be concrete, and in line with our chosen policy on security above, we choose the field $\mathbb{F}_{2^{271}}$, and the elliptic curve whose points are solutions (x, y) over this field to the equation:

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

In this case the number of points on the curve is known to be $2^{271} + 2^{136} + 1 = 487805 \cdot r$ where r is a large prime, large enough to make any Pohlig-Hellman-like attack [15] on the ECDLP problem infeasible. However we must also consider the security of the discrete logarithm over $\mathbb{F}_{(2^{271})^4}$, where it is vulnerable to an index calculus type of attack. The current state of knowledge of the DLP over extension fields is not well studied. However it is believed to be roughly the same as that of the DLP over $\mathbb{F}_{2^m}$, for prime m, where $m \approx 4 \cdot 271 = 1084$. In this setting the current record is for $m = 613$ [19], so we have (for now) a relatively wide margin of safety.

On these curves the fastest known pairing is the η_T pairing [2]. This pairing can be evaluated very efficiently. All such algorithms consists of two parts, a Miller loop, followed by a final exponentiation. For the η_T pairing the loop is truncated to half the length of the related η pairing (which is in turn closely related to the Tate pairing), and the final exponentiation cost is negligible.

The cost of mapping identities to curve points should also be considered. A simple method is to hash the identity to the x-coordinate, and then solve a quadratic equation to find y (see [15] for a fast algorithm). If the quadratic equation has no solution, increment x and try again. Otherwise pick one of the two solutions for y and multiply the point (x, y) by the cofactor, in this case 487805, to obtain a point of order r. This will be a negligible computation compared to the time required to calculate the pairing.

The non-deterministic nature of this process is nevertheless a little unsatisfactory, so recall that in this setting identities can be assigned to the nodes by the deployer of the WSN. So the deployer might only assign those identities for which the quadratic equation is known to have a solution, at the negligible cost of adding approximately 1 bit to the overall length of the node identities.

2.2 Non-supersingular pairing-friendly Elliptic curves

There is an alternative to the supersingular elliptic curves, with their limited choice of embedding degree k. Special families of non-supersingular elliptic curves $E(\mathbb{F}_p)$ can also be pairing friendly [10], and the Tate pairing now evaluates as an element in $\mathbb{F}_{p^k}$, for some reasonable value of k. In fact the condition for being pairing-friendly with an embedding degree of k, is that k is the smallest positive integer such

that $r|(p^k-1)$, and there is no reason that non-supersingular curves cannot be found to meet this condition. These pairings are sometime called "type 3" pairings (whereas pairings on supersingular curves are "type 1") [12]. These have all the properties of type 1 pairings: except that of symmetry $e(A,B) \neq e(B,A)$. As it turns out this has interesting implications which we will return to later. For the moment it is enough to state that the lack of this feature is not crucial, and type 3 pairings can also be used for our purposes.

There is another significant feature of type 3 pairings that must be mentioned. Whereas one parameter of the pairing may be a point on $E(\mathbb{F}_p)$, the best that can be done for the other is that it should be a point on $E'(\mathbb{F}_{p^d})$, where d is a divisor of the embedding degree k, and E' is some twist of the original elliptic curve.

From among the many possible families of pairing friendly non-supersingular elliptic curves [10], here we choose an MNT pairing-friendly curve [30] with $k = 4$:

$$y^2 = x^3 - 3x + B$$

The prime p is chosen to be 160 bits in length, and the number of points on the curve is $34 \cdot r$, where r is a large prime. To be specific:

$$\begin{aligned} p &= \texttt{E3F367D542C82027F33DC5F3245769E676A5755D} \\ B &= \texttt{DABC0397E45200C4DF4CF67714DB64EB866BA034} \\ r &= \texttt{6B455E0A014F1E30EAEF7300BD4BB4258290FC5} \end{aligned}$$

Since r is 155 bits in length, there is adequate resistance to a Pohlig-Hellman type of attack on the discrete logarithm problem [15], which would require around 2^{78} steps to succeed. However the resistance to an index calculus attack [15] is more problematical. Given that the embedding degree is 4, the number of "bits of DLP difficulty" is $4 \cdot 160 = 640$. The current record is 531 bits [22], so again we can consider ourselves safe for the moment. And the level of security here can be considered similar to that for our supersingular curve, given the current state of knowledge [24].

The pairing we recommend here is the original Tate pairing. For this pairing the first parameter to the pairing can be a point on $E(\mathbb{F}_p)$, and the second parameter can be a point on the quadratic twist $E'(\mathbb{F}_{p^2})$.

To map identities to curve points, again hash the identity to x and attempt to solve a quadratic equation for y. Again only identities that are known to have solutions can be chosen. In this case the process is a little more time-consuming, requiring the calculation of a modular square root. Furthermore there is a requirement to hash identities to points of order r on $E(\mathbb{F}_p)$, which requires a multiplication by the cofactor 34. For the second parameter to the pairing, a multiplication by a larger cofactor would be necessary to map to a point of order r on $E'(\mathbb{F}_{p^2})$. However here we can exploit the property of the Tate pairing that its second parameter need not be of order r for bilinearity to hold, and so this potentially expensive operation can be avoided [37]. In fact it is to benefit from this optimisation that we recommend the Tate rather than the Ate pairing [17]: a truncated loop variant of the Tate pairing (which is marginally faster, but requires a first parameter on $E'(\mathbb{F}_{p^2})$ of order r).

3. APPLICATIONS TO WSN SECURITY

Having selected our pairings, let us consider again the key establishment protocol described above – the Sakai-Ohgishi-Kasahara protocol [36]. This protocol was independently rediscovered by Dupont and Enge [8], who also established its security, and considered its implementation on both supersingular and ordinary curves. On supersingular curves, the protocol is as described above. However on ordinary curves, due to the loss of symmetry, a somewhat more elaborate protocol appears to be required. Here we follow Dupont and Enge (with a minor modification).

As before the trusted authority assigns identities and secrets to each node. The node identities – perhaps just a simple serial number – are then mapped to points on the elliptic curve. The node represented by the identity ID_A is hashed and mapped to a point on $E(\mathbb{F}_p)$ of order r via $A_{sr} \leftarrow 34 \cdot H_1(ID_A)$, and to a point on $E'(\mathbb{F}_{p^2})$ via $A_{sk} \leftarrow H_2(ID_A)$. Here H_1 and H_2 are functions which hash arbitrary strings to points on the elliptic curves $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^2})$ respectively. Node A is then issued with the pair of secrets sA_{sr} (its "source" secret) and sA_{sk} (its "sink" secret). If a node B is similarly equipped, then they can each calculate a mutual key as

$$H(e(sA_{sr}, B_{sk})e(B_{sr}, sA_{sk})) = H(e(sB_{sr}, A_{sk})e(A_{sr}, sB_{sk}))$$

where A calculates the left-hand-value, and B calculates the right-hand-value. They are clearly the same because of bilinearity. Note that there are fast ways to calculate products of pairings as required here [37].

The extra complication is due to the necessity to restore symmetry to the calculation. However if a small amount of interaction is allowed prior to the establishment of the key, then a simpler protocol can be used. If the nodes can agree that one will use its source key, and the other its sink key, then a mutual key can be determined as

$$H(e(sA_{sr}, B_{sk})) = H(e(A_{sr}, sB_{sk}))$$

Note that if both sides will use the same type of key (sink or source key) they will not be able to establish a mutual key. One of the nodes must use its source key and the other its sink key. Only different types of key can be used, otherwise the pairing cannot be calculated [13].

As we can see a key exchange protocol based on the type 3 pairing is a little bit more complicated than the simple Sakai-Ohgishi-Kasahara scheme. It would seem that a type 1 pairing should be preferred over a type 3 pairing for the majority of WSN applications. Type 1 pairings are also much faster to compute, as we will show later. Nevertheless, a case can be made for the use of the type 3 pairing, and is given below.

3.1 Resistance to node compromise attacks

If a node is captured by an attacker, its secrets can be extracted, and the node converted to meet the needs of its new owner. There is not much that can be done to prevent this. One simple idea would be that a node – just before its battery expires – should delete its keys. A "dead" node should reveal no secrets.

Each node in a WSN will act as a source of data, and perhaps also as a sink for data. It is envisaged in fact that most nodes might be pure sources, most of the time, and that relatively few would be sinks. One plausible scenario is that a group of nodes might elect to send all their data to a single sink node, who might then collate the data before transmission back to the WSN owner. This role as data collator might be passed around, as the selected node will be running its battery down due to transmission of large amounts of data. Many routing algorithms for WSNs (e.g [16], [25]) use such clustering and data aggregation techniques.

A particularly insidious attack is for a subverted node to announce itself as a data sink, hence collecting data from the WSN and passing it on to its new master. This suggests the following strategy: A node whose battery is running low and who is unlikely (due to its low battery or its physical placement within the network) to be required to operate as

a data sink, should then delete its own sink key. Now the captured node can only operate as a source of data, and so is likely to be of much less use to its new owner.

Alternatively most of the nodes might be designated source-only, with maybe only a small proportion being sink-enabled. Therefore we suggest that type 3 pairings may be of more value in high security WSNs, as they offer some degree of extra protection against node compromise. Clearly the capture of a type-3 node with source-only capabilities, will result in less damage than the capture of a node which can masquerade as either a legitimate data source or as a legitimate data sink.

4. TYPE 1 PAIRINGS OVER $\mathbb{F}_{2^M}$

As described in section 2.1 we have implemented the η_T algorithm to calculate the pairing on a supersingular elliptic curve. Those kinds of curves lead to more efficient implementations in terms of bandwidth, memory usage and processing speed than any other pairing algorithms for ordinary curves. All these features make this type of pairing especially attractive for low power environments like wireless sensor networks where available resources are very constrained.

Algorithm 1 Computation of $\eta_T(P,Q)$ on $E(\mathbb{F}_{2^m}) : y^2 + y = x^3 + x$

INPUT: P, Q
OUTPUT: $\eta_T(P,Q)$
1: **let** $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$
2: $u \leftarrow x_P + 1$
3: $f \leftarrow u \cdot (x_P + x_Q + 1) + y_P + y_Q + 1 + (u + x_Q)s + t$
4: **for** $i \leftarrow 1$ **to** $(m+1)/2$ **do**
5: $\quad u \leftarrow x_P$, $x_P \leftarrow \sqrt{x_P}$, $\quad y_P \leftarrow \sqrt{y_P}$
6: $\quad g \leftarrow u \cdot (x_P + x_Q) + y_P + y_Q + x_P + (u + x_Q)s + t$
7: $\quad f \leftarrow f \cdot g$
8: $\quad x_Q \leftarrow x_Q^2$, $\quad y_Q \leftarrow y_Q^2$
9: **end for**
10: **return** $f^{(2^{2m}-1)(2^m - 2^{(m+1)/2}+1)}$

Algorithm 1 (based on [2]) presents an optimized scheme for the calculation of the η_T pairing. In fact it is a variant of the basic Miller algorithm for computing the pairing. Variables P and Q are two points on a supersingular elliptic curve over $\mathbb{F}_{2^m}$. Pairs (x_P, y_P) and (x_Q, y_Q) are simply the coordinates of the points P and Q respectively. The finite field $\mathbb{F}_{2^m}$ is often called a *binary finite field* and can be represented over a polynomial or normal base. In our approach we use the polynomial base where any given element $a(x) \in \mathbb{F}_{2^m}$ can be represented as a binary polynomial with at most $m-1$ coefficients $A_i \in \{0,1\}$:

$$a(x) = A_{m-1}x^{m-1} + \cdots + A_2x^2 + A_1x + A_0$$

The field element $a(x)$ can be then simply denoted as a bit string $(A_{m-1} \ldots A_2 A_1 A_0)$. Arithmetic on such elements is performed exactly as on standard polynomials with integer coefficients. Values f and g in Algorithm 1 are elements evaluated in the extension field $\mathbb{F}_{2^{4m}}$ and they can be efficiently represented as polynomials with 4 coefficients in $\mathbb{F}_{2^m}$. So for example element $b(z) = B_3z^3 + B_2z^2 + B_1z + B_0$ in $\mathbb{F}_{2^{4m}}$ can be denoted as a vector $[B_3, B_2, B_1, B_0]$. Elements $s, t \in \mathbb{F}_{2^{4m}}$ in our implementation of the η_T pairing are equal to $[0,1,1,0]$ and $[0,0,1,0]$ respectively. Those values were derived from the distortion map which exists only on supersingular curves [29]. In our case the distortion map maps a point from $E(\mathbb{F}_{2^m})$ to $E(\mathbb{F}_{2^{4m}})$.

Looking at Algorithm 1 we can see that the most expensive part is the `for` loop which needs to be executed $(m+1)/2$ times. All the operations inside this loop have to be optimized in terms of execution speed in order to achieve a good performance of the algorithm. The most time consuming operation inside this loop is the polynomial multiplication and its implementation has a major impact on the overall execution time of the η_T pairing. In particular multiplication of $\mathbb{F}_{2^{4m}}$ values (line 7) is very time consuming but consists of multiplications in $\mathbb{F}_{2^m}$ which again emphasizes the importance of base field multiplication. Using the Karatsuba method [23] for $\mathbb{F}_{2^{4m}}$ multiplication decreases the number of necessary operations to 9 modular multiplication and some additions which are very cheap in binary fields (just bitwise exclusive-or of the coefficients of the polynomials). Inside the Miller loop there are also other important operations like squaring and calculation of the square root of the field element. However those functions are not as complex as binary polynomial multiplication and when efficiently implemented can be performed much faster.

The last stage of the η_T algorithm is the final exponentiation. This step is not very time consuming as it can be performed for the relatively inexpensive cost of $(m+1)/2$ extension field squarings, four extension field multiplications, one extension field division and some other cheaper arithmetic operations.

4.1 Implementation details

The total cost of the η_T pairing ($m = 271$) in terms of basic arithmetic operations is given in Table 1. Additions are not taken into consideration because they are fast operations in $\mathbb{F}_{2^m}$. The bottom line is that the key to efficient η_T implementation lies in the performance of binary polynomial multiplication. That is the main reason why, in our implementations for WSN platforms, we decided to use assembly language for all the basic polynomial arithmetic.

Table 1: Cost of the η_T pairing on $y^2 + y = x^3 + x$ curve

	ModMuls	ModSquares	SqRoots
Miller Loop	1904	544	544
Final Exp.	114	139	0

Our work was focused on a wide range of available processors that are commonly used in WSNs. The most popular platform is MICA2/MICAZ which embeds the 8-bit ATmega128L MCU [1] from Atmel. Our second target platform was Tmote Sky produced by the Moteiv company. This WSN node uses the Texas Instruments MSP430F1611 chip [45]. We have also prepared an implementation of our pairing programs for the PXA271 XScale microprocessor [18] which is based on an ARM core. This MCU is being used in recent Imote2 nodes, which have more computing power and a much improved capabilities than the previously mentioned platforms.

Our pairing implementations are based on the MIRACL [39] library which provides all the necessary tools to perform operations on elliptic curves. It supports Elliptic Curve Cryptography over both standard finite fields $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. It also handles well all the big integer and large polynomials arithmetic. All memory allocation in our programs was taken directly from the stack. This means that after the pairing calculation almost all of the RAM memory could be re-used for different purposes. Our main task was to optimize the MIRACL library to work on three completely different 8, 16 and 32-bit architectures. Each of those processors has specific features that needed to be exploited in order to achieve maximum efficiency. Our results show that even such a complex operation as a pairing can be quickly

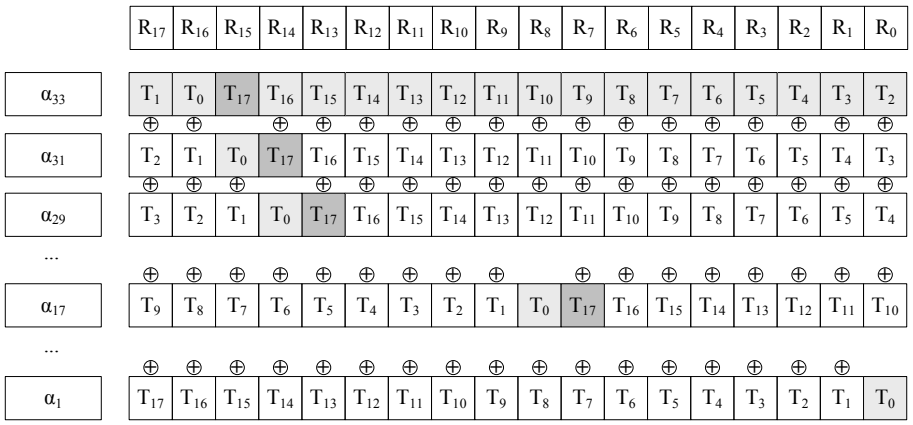

Figure 1: First part of the multiplication $a_1(x)b_1(x)$. T0 to T17 denote the 18 bytes of the precomputation table according to $\alpha_{2i+1}b(x)$. The bytes without a filling are added to the registers. Bytes filled with light-gray are loaded into the registers. The dark-gray bytes are stored in the RAM after the addition (except T17 of Hi(Byte 16)).

and efficiently calculated even on a very constrained architecture like the ATmega128L.

According to our chosen policy on security (section 2.1) parameter m was set to 271 and the embedding degree k was 4. The fact that we were working with a fixed field size allowed us to greatly optimize our code. We used mainly C code and some assembly language to speed up our time critical arithmetic routines. Operations like modular reduction and calculation of the square root were strictly optimized for a specific form of reduction polynomial $f(x) = x^{271} + x^{201} + 1$. It turned out in our experiments that field-specific assembly code gives the best performance for the η_T pairing algorithm. The following sections describe in more detail the implementations for our three target platforms.

4.2 ATmega128L 8-bit processor

The ATmega128L is a modern RISC processor which offers a variety of different instructions, most of which are executed in a single CPU cycle. There are 32 available 8-bit registers which gives the developer a lot of flexibility when writing the source code. The Atmel product offers 128KB of ROM and only 4KB of RAM which can be a problem in more demanding applications. Assembly code development on this platform can be quite challenging but, as we will show later in this section, it can also lead to nice improvements both in terms of execution time and memory usage.

As we pointed out earlier, efficient arithmetic in the binary field has a major influence on η_T pairing performance. This section will focus on optimizations made for the AVR 8-bit architecture.

For our η_T pairing all the elements of the finite field are represented by polynomials defined over $\mathbb{F}_2$ of degree less than or equal to 271. For the internal representation of the polynomials we use byte arrays of length 34. There are four critical operations in the finite field, which have to be optimized: multiplication, squaring, square root computation and modular reduction. All these operations are realized in assembly language routines.

The simplest way to perform polynomial multiplication is the "shift-and-add" method but it is usually slow when implemented in software because of many vector shifts. Such a method is more suited for hardware rather than software especially on a small 8-bit processor, where shifts can only by performed 1 bit at a time. Also in modern architectures there is no direct counterpart to the integer multiplication instruction when it comes to binary polynomial multiplication. Considering all the constraints of the 8-bit platform we decided to use a hierarchical approach.

The multiplication of binary polynomials of degree less than 271 is performed in two steps and results in a polynomial of degree at most 540. The first step is the application of Karatsuba's method [23] substituting one full multiplication by three half-size multiplications: The two factors $a(x), b(x) \in \mathbb{F}_{2^{271}}$ are split into smaller polynomials $a_1, a_2, b_1, b_2 \in \mathbb{F}_{2^{136}}$ with $a(x) = a_1(x)x^{136} + a_2(x)$ and $b(x) = b_1(x)x^{136} + b_2(x)$. Then the multiplication $c(x) = a(x)b(x)$ leads to the formula:

$$c(x) = a_1b_1x^{272} + ((a_1+a_2)(b_1+b_2) + a_2b_2 + b_1a_1)x^{136} + a_2b_2$$

The time consuming part is the computation of the subproducts a_1b_1, a_2b_2 and $(a_1 + a_2)(b_1 + b_2)$. In order to decrease the computation time as much as possible, we have implemented a comb method of window width 4 (see [15]) in the second step of the multiplication. For this purpose we write the factor $a_1(x)$ in the following way:

$$a_1(x) = \sum_{i=0}^{135} A_i x^i = \sum_{i=0}^{33} \underbrace{\left(\sum_{j=0}^{3} A_{4i+j} x^j \right)}_{\alpha_i} x^{4i}$$

The comb multiplication with windows of width four is then executed in two loops. In the first loop all multiples $\alpha_{2i+1}b(x)x^{8i}$, $0 \le i \le \lceil (d-1)/4 \rceil$ are summed up in the polynomial $c(x)$. Then $c(x)$ is multiplied by x^4 and finally the polynomials $\alpha_{2i}b(x)x^{8i}$, $0 \le i \le \lceil (d-1)/4 \rceil$, are added to $c(x)$ again. The multiples $u(x)b(x)$, $\deg(u) < 4$ can be precomputed and stored in a lookup table. This table requires 288 bytes.

The most important problem, which impacts significantly on the performance of the multiplication, is the handling of the polynomial $c(x)$. Clearly, temporary result cannot be kept in the CPU registers for the entire duration of the multiplication. Actually $c(x)$ is rotated through the available registers of the microcontroller, such that only the 18 bytes which are affected by the addition of $u(x)b(x)$ are stored in the registers (see Figure 1). The rotation minimizes the number of memory accesses, but all loops have to be unrolled, and so the code size increases.

Furthermore we have implemented a particular squaring routine, since squaring of a polynomial defined over a binary field is much simpler than standard multiplication. The result of squaring of a polynomial is obtained by "spreading" its coefficients and inserting zeros in between: $a^2(x) = \sum_{j=0}^{270} A_j x^{2j}$. This spreading can be optimized using a lookup table for small polynomials. We based the implementation on a table which maps four bits to one byte. This table needs 16 bytes of RAM.

In the finite field $\mathbb{F}_{2^{271}}$ we have to make all the arithmetic modulo the irreducible polynomial $f(x) = x^{271} + x^{201} + 1$, so after multiplication and squaring the result has to be reduced. To compute the representative of a polynomial of degree less than 541 we reduce eight coefficients stored in one byte by the congruence $x^{272} \equiv x^{202} + x$. The function is similar to the reduction technique for the NIST parameters proposed in [15]. Then one byte can be reduced with four shift and four addition operations as well as four memory accesses.

Square roots in $\mathbb{F}_{2^m}$ can be implemented pretty efficiently when we can optimize the procedure for a certain reduction polynomial [9]. For a polynomial $a(x) = \sum_{j=0}^{270} A_j x^j$ we have:

$$\sqrt{a(x)} = \underbrace{\sum_{j=0}^{135} A_{2j} x^j}_{a_1} + \sqrt{x} \underbrace{\sum_{j=0}^{134} A_{2j+1} x^j}_{a_2}$$

Similar to the method in [40], the shrinked polynomials a_1 and a_2 are obtained by some table lookups. The crucial point of the square root computation is the multiplication of a_2 by $\sqrt{x}$. As it is pointed out in [40] the square root follows by the congruence:

$$x^{272} + x^{202} = x$$

Since all non-zero exponents of the reduction polynomials are odd, the square root of x equals to $x^{136} + x^{101}$. Finally the combination of a_1 and a_2 is performed by three additions of polynomials.

4.3 TI 16-bit MSP430 processor

Our second target processor was a 16-bit MSP430F1611 from Texas Instruments. This platform differs in many ways from the Atmel chip. The MSP430 has a more traditional architecture and uses mainly memory based operations. Its instruction set is limited to 27 instructions but the variety of 7 different addressing modes offers a lot of flexibility in data manipulation. This CPU offers also 10KB of RAM but only 48KB of ROM which might not be sufficient for large programs.

It may seem that an implementation of the Toom-Cook method [5] would be appropriate for polynomials multiplication in characteristic 2 on this type of hardware architecture. However according to [5] this technique is faster than Karatsuba only for polynomials above the degree of 352. In our case of $\mathbb{F}_{2^{271}}$ it would give us lower performance. Finally we implemented a one stage approach for calculation of binary polynomials multiplication. We used the comb method with window width of 8 instead of 4 because shifting by a byte can be performed more efficiently on the MSP430 architecture. Although the obtained results were quite good in terms of speed, the increase in memory space for our assembly routines was really significant. The hierarchical approach on the other hand can save on memory by using the "divide-and-conquer" technique with consecutive calls to the same multiplication routine.

Eventually we decided to use a similar two stage technique as for the ATmega128L. The main problem in the MSP430 case was that we could only use 12 available registers instead of 32. We had to divide one 17 word polynomial into two smaller 9 word polynomials in order to apply the Karatsuba method (an extra last word containing all zeros for the second polynomial had to be added to maintain the symmetry). Then comb multiplication was applied with window width of 4 to calculate the partial products. A lookup table of 320 bytes had to be precomputed for each partial product. We were using 16-bit words and 4 bits scanning, so we had to perform the calculations in 4 loops multiplying $c(x)$ by x^4 in between the loop iterations. The temporary result was held in 10 registers and the remaining 2 were used to perform all the necessary operations. Rotation of necessary words was performed in a similar manner as in Figure 1.

We could take advantage of 16-bit processing in case of the MSP430 for the squaring routine in $\mathbb{F}_{2^{271}}$. We used an algorithm that scans eight bits of the polynomial and maps them into 16 bits of the result. To optimize this process, a lookup table of size 512 bytes was precomputed. Square root computation was implemented according to [40] with minor tweaks to facilitate operations on 16 bits words. We used a similar technique for reduction as in algorithm 2.42 in [15].

4.4 Intel 32-bit PXA271 processor

We implemented our software on a low-power 32-bit Marvell PXA271 CPU which is also used in wireless sensor nodes. This architecture is built around the ARM core and differs completely from the two previously described platforms. It is a much more advanced and powerful CPU with superior capabilities. This range of microprocessors is commonly used in smartphones (e.g. iPhone) and PDA class devices and opens new possibilities when used in the WSN environment.

The PXA271 offers full 32-bit processing and embeds also an MMX DSP coprocessor for multimedia applications. The clock rate on this Xscale CPU can be dynamically changed by adjusting the input voltage. This feature allows us to precisely set the tradeoff between energy consumption and computational power during CPU operation. Also the memory resources on this CPU are a few magnitudes higher than on the other two platforms. The Xscale platform uses ARM instruction set version 5TE [42] which is very user friendly and allows efficient data processing. The PXA271 has a standard RISC load-store architecture, with additional features, like shifting of operands at no extra cost (using a built-in barrel shifter) and conditional execution of instructions. We have 16, 32-bit registers available for general use, although three of them are reserved for special purposes like stack pointer (R13), program counter (R15) and function return address (R14). If needed the content of R14 can be pushed onto the stack and an extra register is then available until the end of the called function.

Most of the algorithms for binary polynomial arithmetic are optimized with a 32-bit platform in mind. This fact allowed a straight-forward implementation on our PXA271 CPU. We did not have to use a two stage approach in the case of polynomial multiplication in $\mathbb{F}_{2^{271}}$ on this 32-bit platform. The whole polynomial fits in 9 registers, so using Karatsuba here would only result in additional overhead. We used the comb method straight away with a window size of 8. Scanning was performed 2 bits at once so the amount of required precomputation was exactly the same as it would be with window size 4 (576 bytes). This modification decreased the number of necessary loop iterations from 8 to 4 and decreased the number of polynomial shifts of $c(x)$ from 7 to 3. These savings were made at a cost of

Table 2: Timings in clock cycles for modular arithmetic routines and pairing operations in $\mathbb{F}_{2^{271}}$ on different WSN processors

	Atmega128				MSP430				PXA271			
	Mul	**Sqr**	**Sqrt**	$\eta_T(P,Q)$	**Mul**	**Sqr**	**Sqrt**	$\eta_T(P,Q)$	**Mul**	**Sqr**	**Sqrt**	$\eta_T(P,Q)$
Assembly	13557	1581	1730	19,660,993	10147	1363	1644	14,097,304	4926	499	546	6,002,134
C code	66271	4711	12021	80,608,843	40666	3667	11212	50,684,686	13183	2375	2496	16,974,044
Decrease	80%	66%	86%	76%	75%	63%	85%	72%	62%	79%	78%	65%

Table 3: Memory requirements for the η_T pairing operation on different WSN processors

	Atmega128		MSP430		PXA271	
	ROM	**Stack**	**ROM**	**Stack**	**ROM**	**Stack**
Assembly	47.41KB	3.17KB	23.66KB	4.17KB	29.55KB	4.12KB
C code	41.23KB	3.17KB	23.01KB	4.17KB	37.24KB	4.12KB
Increase	15%	0%	3%	0%	-20%	0%

extra polynomial additions and table lookups in the main step of the algorithm. Temporary result was held in 9 registers while the remaining 5 were used for storing operands addresses and choosing the right lookup table element. All loops were unrolled to achieve further savings in execution time. Shifting the temporary result vector by 8 bits instead of 4 allowed us also to better optimize this operation for 32-bit architecture. We used one loop and 2 pointers that were simply loading and storing appropriate bytes. Free shifts and multiple load/store instructions on the PXA271, allowed us to achieve far better performance than on two previous platforms.

4.5 Type 1 pairing results

Table 2 shows cycle accurate timings that we have achieved on different WSN processors. All the modular arithmetic routines and pairing operations were performed in $\mathbb{F}_{2^{271}}$. We used AVR Studio 4.13 for the Atmega128L processor, IAR Embedded Workbench v4.10 for the MSP430 and IAR Embedded Workbench v5.0 for the ARM platform, to obtain all the timings in Table 2. Results achieved with our assembly routines are compared with C only implementation to show the savings in execution time.

We have noticed significant differences for the same optimization levels when different compilers were used (for e.g. gcc and IAR compiler). That was the main reason, why we used the same settings for the compilers in all cases. Optimization flag -O0 was set during all simulations, so our results can be directly compared with other implementations no matter which compiler is used.

As we can see the difference between our standard C code functions and specially optimized assembly routines is quite significant. All the operations execution times were decreased by around 60 to 85%. Handcrafted code gave us even better performance than we had expected. Square root computation was around 4-7 times faster and, (of the most importance for the η_T algorithm), polynomial multiplication was as much as 5 times faster. Using all these specially optimized arithmetic routines, we were able to calculate the η_T pairing 65-76% quicker. In the best case (ATmega128 CPU) it was even 4 times faster than C only implementation. It appears that careful optimization of critical routines in assembly language leads to a big performance increase on these embedded microcontrollers. Usually on standard desktop computers savings of around 20-30% are possible to achieve when using assembly language.

Results presented in Table 2 are especially significant because in almost all cases we managed to keep the same level of memory usage. The memory requirements for the η_T pairing on our three research platforms are presented in Table 3. Stack usage in all implementations remained on the same level as our assembly routines did not use additional variables. RAM utilization may seem high but the memory is reserved only for the duration of pairing calculation. After that all of that RAM memory is released and can be reused for different purposes. Stack size values presented in Table 3 were also the peak numbers during program execution. Average stack utilization was usually much lower than that.

The increase in memory overhead is considerable only on the ATmega128 platform but gives the best performance results. For the MSP430 processor the 3% increase in ROM utilization is negligible, as it leads to 72% improvement in execution time. On the PXA271 microcontoller we were even able to achieve a 20% decrease in program space.

5. TYPE 3 PAIRINGS OVER $\mathbb{F}_P$

Our implementation of the Tate pairing is of the algorithm as described in [38], and we exploit a fast trick for the final exponentiation (see [38] section 10.3.1). An implementation in C can be found in the file ake4mntt.c in [39]. For convenience we first precalculate the constant

$$\delta \quad = \quad (p^2+1)/(34r) - p = \texttt{F19192168B16C1315D34}$$

Algorithm 2 Computation of $e(P,Q)$ with Tate pairing

INPUT: $P \in E(\mathbb{F}_p), Q \in E'(\mathbb{F}_{p^2}), \delta$
OUTPUT: $e(P,Q)$
1: $T \leftarrow P$, $f \leftarrow 1$
2: $s \leftarrow r-1$
3: **for** $i \leftarrow \lfloor \lg(s) \rfloor - 1$ **downto** 0 **do**
4: $\quad f \leftarrow f^2.l_{T,T}(Q)$
5: $\quad T \leftarrow 2T$
6: $\quad$ **if** $s_i = 1$ **then**
7: $\quad\quad f \leftarrow f.l_{T,P}(Q)$
8: $\quad\quad T \leftarrow T+P$
9: $\quad$ **end if**
10: **end for**
11: $f \leftarrow f^{p^2-1}$
12: $f \leftarrow f^{34}$
13: $f \leftarrow f^p.f^\delta$
14: **return** f

The line functions $l_{T,T}(Q)$ and $l_{T,P}(Q)$ are calculated as described in [38], although we use ideas from [6] to recycle calculations from the elliptic curve point doubling and point addition formulae. As an effect of these steps the calculation of lines 4 and 5, and lines 7 and 8 are combined. Note that

Table 4: Cost of the Tate pairing on MNT $k = 4$ curve

	ModMuls	ModAdds	ModInvs
Miller Loop	5730	15006	2
Final Exp.	571	2642	1

the calculation of f^p is effectively for free, using the frobenius action. So for example the calculation of f^{p^2-1} requires only two applications of the frobenius, and a single inversion. In practise we also use a windowing method for the main Miller loop, but omit details here in the interests of clarity.

The most time consuming operation is that of modular multiplication (ModMul), and to a lesser extent of modular addition or subtraction (ModAdd). Modular inversion (ModInv) is expensive, but rare. For maximum efficiency we use the Montgomery representation of numbers modulo p [31], so that modular reduction requires no divisions. Our code for multiplication, reduction, addition and subtraction is all in the form of automatically generated loop-unrolled assembly language. The generic method used requires only that the processor supports an unsigned integer multiplication instruction, has some form of indexed addressing, and can handle add-with-carry operations. For multiplication (and squaring) we use the variation of the classic Comba method as described in [41]. We use stack allocation for all variables, and some effort was made to keep code size to a minimum by removing unnecessary functions.

The overall cost of the pairing in terms of the primitive operations is given in Table 4.

5.1 ATmega128L 8-bit processor

Although efficient elliptic curve arithmetic is a very important issue while implementing pairing based cryptography it will not be our primary concern here. In this section we will focus on the implementation of basic arithmetic routines over $\mathbb{F}_p$ as they have the biggest impact on the Tate pairing performance. Multiprecison integer multiplication will be our priority function due to the fact that this is the most time critical routine that is repeated over 6000 times throughout the pairing calculation (see Table 4).

On an 8-bit processor, values in $\mathbb{F}_p$ are naturally represented as 20-byte arrays. As we stated earlier, this Atmel chip supports a nice RISC instruction set, which in particular implements an 8×8 bit integer multiplication instruction which can be executed in just 2 clock cycles. The hybrid method [41] for integer multiplication on Atmega128L makes good use of all 32 available registers to give a significant speed up for the Tate pairing algorithm. Modular inversion is a very expensive operation on the ATmega128L but luckily it has to be performed only three times during the pairing calculation.

5.2 TI 16-bit MSP430 processor

The Tate pairing was also implemented on the MSP430 processor from Texas Instruments. This CPU embeds a 16×16 bit hardware multiplier. If we want to achieve the best performance for multiprecison multiplication on this platform it is important to use this multiplier unit. This device is a peripheral and is not implemented in every member of the large MSP430 family of microprocessors. The MSP430F1611 processor used in Tmote Sky nodes is equipped with a hardware multiplier and it gains access to it through memory mapped I/O registers.

In order to take the full advantage of 16-bit processing on the MSP430 we had to develop all the basic arithmetic routines over $\mathbb{F}_p$ in assembly language. The most important operation – multiplication – was implemented using a hybrid method with column size $d = 2$. This was due to the fact that we could only use 12 available registers (instead of 32 as for the ATmega128L). The Tate pairing evaluates as an element of an extension field $\mathbb{F}_{p^k}$. In our case $k = 4$, so special functions for multiplication, addition, exponentiation and inversion in $\mathbb{F}_{p^4}$ were also developed.

5.3 Intel 32-bit PXA271 processor

The PXA271 is an ARM based processor and code development for this platform was a little bit easier than for the other two devices. Large amount of memory, fast 32-bit processing and rich instruction set allow efficient implementation of complex cryptographic algorithms. The Xscale platform handles well all the multiprecision arithmetic on big integers. In particular the ARM instruction set v5TE implements a 32×32 bit unsigned integer multiplication instruction which can be executed in 3-6 clock cycles depending on the size of the input values.

Although multiprecision arithmetic implementation in C on a PXA271 gives relatively good timings we can still gain on performance by using specially designed assembler code. For the multiplication routine we used a variant of the Comba method with column size $d = 2$. Again this was limited by the number of available registers. A detailed description of this method for the ARM architecture can be found in [41].

5.4 Type 3 pairing results

Our type 3 pairing implementations were tested in the same simulating environments as mentioned in section 4.5. All the evaluation results are presented in Table 5. Timings for modular arithmetic routines are in clock cycles and they assume 160-bit integers in $\mathbb{F}_p$. The number of clock cycles for modular addition is an average value because the reduction step is not always executed. Our results show that field-specific assembly code gives the maximum speed up for the Tate pairing algorithm.

Looking at Table 5 we can see that inversion is the most expensive arithmetic operation on all platforms, but it is not critical as it is performed very rarely. Modular multiplication of big integers is the most important operation that has a big influence on the overall pairing performance. That is why we were putting the most effort in optimizing this operation on our different research platforms. We were able to achieve even up to 66% improvement on this operation with our assembly routines when compared with C only implementation. On Atmega128 the hybrid multiplication with column size $d = 4$ gave us a times 3 improvement whereas on the MSP430 the same method with $d = 2$ was more than twice as fast as the standard multiplication technique. The differences in timings between assembly and C routines on PXA271 were not as significant as it was for the other two platforms. The ARM architecture was initially designed for fast arithmetic operations and there is not much space for improvements when using hand-crafted assembly code. Despite that we could still achieve a 42% improvement in execution time for modular multiplication on this platform.

Table 5 shows also the overall Tate pairing results. Introduction of assembly arithmetic routines leads even up to 62% reduction in execution time. The timings for the Tate pairing calculation were significantly slower than for the η_T algorithm on all processors. On the PXA271 we were able to achieve a 36% improvement which proves that integer arithmetic can be very efficiently executed on an ARM based machine with 32-bit processing even without the use of assembly routines. The difference was much bigger in case of multiplication of binary polynomials.

Memory usage is sometimes more important than execution time and this is certainly the case for small and constrained embedded devices like WSN motes. In our tests we

Table 5: Timings in clock cycles for modular arithmetic routines and pairing operations in $\mathbb{F}_p$ on different WSN processors

	Atmega128				MSP430				PXA271			
	Mul	**Add**	**Inv**	$e(P,Q)$	**Mul**	**Add**	**Inv**	$e(P,Q)$	**Mul**	**Add**	**Inv**	$e(P,Q)$
Asse	7547	404	364291	54,800,077	4734	386	229724	37,739,040	843	155	49223	8,055,473
C	22493	596	419812	143,888,874	11148	533	269768	77,411,534	1463	200	53318	12,573,931
Decr	66%	32%	13%	62%	58%	27%	15%	51%	42%	22%	8%	36%

Table 6: Memory requirements for the Tate pairing operation on different WSN processors

	Atmega128		MSP430		PXA271	
	ROM	**Stack**	**ROM**	**Stack**	**ROM**	**Stack**
Assembly	60.91KB	3.39KB	34.88KB	3.39KB	44.40KB	3.75KB
C code	94.41KB	3.39KB	46.15KB	3.39KB	51.50KB	3.75KB
Decrease	35%	0%	24%	0%	14%	0%

have measured the memory overhead of all our Tate pairing implementations (see Table 6). In all cases we have decreased the code size when introducing assembly language implementations. On the Atmega128 processor our optimized pairing implementation saves over 30KB in comparison with the standard C version. Memory usage results were also quite satisfactory on the MSP430 CPU as we were able to save over 10KB of program space going down from 46.15KB to 34.88KB. This achievement is especially important for the MSP430F1611 platform where only 48KB of ROM are available. Despite these optimization efforts, the memory overhead for the Tate pairing is still quite significant on these devices. Looking from the memory point of view type 1 pairings should be recommended over type 3 pairings on the most limited WSN platforms. Nevertheless we are sure that further improvements in code size for the Tate pairing implementations are possible.

6. RELATED WORK AND OVERALL RESULTS

So far there has been a lot of different security proposals for wireless sensor networks that were focusing on symmetric encryption schemes [48, 35, 20]. All of these studies assumed that Public Key Cryptography is too heavyweight and not suitable for WSNs.

More recent papers [47, 14] have shown that asymmetric security solutions can also be applied to sensor networks. Results presented in those papers proved that Elliptic Curve Cryptography (ECC) is a more efficient technique than RSA in such a resource constrained environment. Those results provoked a large number of follow-on studies [27, 26, 46, 4] on efficient implementation of ECC primitives on WSN nodes. The latest achievements in this area [44, 43] shows that ECC primitives can be calculated in a quick and efficient way on small 8 and 16-bit platforms.

Pairing based cryptography is regarded as even more complex than ECC. With this work we are going a step forward to show that PBC primitives can be also efficiently implemented on sensor devices. On our most constrained 8-bit MICA2 nodes the fastest pairing takes 2.66s – a time comparable with the point multiplication operation[1]. Our results prove that PBC can be now used as an attractive alternative to ECC in many different security schemes for sensor networks.

Although much research has been carried out on pairings on standard desktop class computers, not much attention has focused on implementing those complex operations on small and constrained devices. In the literature there are some papers [7, 32] that envisage WSNs as a good application space for PBC. However no implementation on a sensor node was presented. Few results are available concerning the practical deployment of PBC in WSN motes. The first pairing implementation on a sensor node was performed by Oliveira *et al.* [34]. The Tate pairing over $\mathbb{F}_p$ with $k = 2$ and p a 256-bit prime was implemented on a MICAz mote which uses the ATmega128L MCU. The level of security proposed in that work is below the current record for breaking the DLP, so this implementation cannot be used in all applications. Also the timings reported in [34] are far from being optimal. Much better results were presented in [44] and [33]. Especially the work of Szczechowiak *et al.* [44] is significant because it implements PBC over both $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ on two widely used sensor node platforms (MICA2 and Tmote Sky).

This work presents the first in-depth study on the application of pairing-based cryptography to wireless sensor networks. We took a close look at implementation issues of PBC primitives on a broad range of WSN platforms. We have also considered pairings application on a system level proposing specific protocols for use in a sensor network environment. Our implementations cover typical processors used in WSN nodes like 8-bit Atmega128L, 16-bit MSP430F1611 and 32-bit PXA271.

The overall results for our pairing computations, assuming 7.3828MHz clock rate on MICA2, 8.192MHz on Tmote Sky and two different clock speeds for Imote2, are presented in Table 7. As we can see the pairing calculation can be performed in as little as 2.66s on a tiny 8-bit Atmel CPU. The timings for other more powerful processors are much faster. The η_T pairing over $\mathbb{F}_{2^{271}}$ is significantly quicker than the Tate pairing on all our target platforms. The difference is a lot smaller on the PXA271 processor where arithmetic operations over $\mathbb{F}_p$ can be executed much more efficiently. The program code is also noticeably smaller for the type 1 pairing in all cases. These features favours the usage of the η_T pairing especially on the two more constrained processors where available memory is very limited.

We have also measured the energy consumption of our pairing algorithms on two of our research platforms. An experimental setup was used for the MICA2 and the Tmote Sky motes to measure the current drawn from the batteries during our programs execution. In the case of the Imote2 platform we used the values from the manufacturer data sheet. As we can see in Table 7, the η_T pairing is more efficient in terms of energy consumption on all devices. Due to faster computation time and the use of power management unit, the total energy usage for η_T pairing on Imote2 is the lowest among all the platforms. The core voltage on Imote2

[1]scalar point multiplication is a dominant operation in ECC protocols such as ECDH and ECDSA

Table 7: Overall results for pairing implementation over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ on typical WSN nodes

	MICA2		Tmote Sky		Imote2 (13MHz)		Imote2 (104MHz)	
	$\eta_T(P,Q)$	$e(P,Q)$	$\eta_T(P,Q)$	$e(P,Q)$	$\eta_T(P,Q)$	$e(P,Q)$	$\eta_T(P,Q)$	$e(P,Q)$
Timing	2.66s	7.43s	1.71s	4.61s	0.46s	0.62s	0.06s	0.08s
ROM	47.41KB	60.91KB	23.66KB	34.88KB	29.55KB	44.40KB	29.55KB	44.40KB
STACK	3.17KB	3.39KB	4.17KB	3.39KB	4.12KB	3.75KB	4.12KB	3.75KB
Current draw	7.86mA	7.88mA	3.45mA	3.68mA	31mA*	31mA*	66mA*	66mA*
Energy usage	62.73mJ	175.65mJ	17.70mJ	50.89mJ	12.12mJ	16.34mJ	3.76mJ	5.02mJ

* - manufacturer data

is only 0.85V when clocked at 13MHz with radio transceiver turned off. Tmote Sky node is also very efficient in terms of energy consumption. The Imote2 is surprisingly the most energy efficient device even though it is the most powerful of our research platforms with the highest clock speed. All calculations on Imote2 can be significantly accelerated with the increase of clock speed to 104MHz, which simultaneously decreases the energy consumption.

To our knowledge, the implementations presented in this paper are the fastest yet reported on all three platforms. Figure 2 presents the comparison of our pairing implementations with the research results that were presented in the literature so far. The comparison includes only the MICA2 and Tmote Sky platforms as we are not aware of any pairing implementations for the Imote2 mote. The Tate pairing on MICA2 in our implementation takes only 7.43 seconds to complete when clocked at 7.3828MHz. This compares very favourably with the timings of Tinytate [34] (30.21s) and NanoECC [44] (17.93s). The difference (over 10 seconds) is pretty significant especially when we will take into consideration the mote's limited energy resources. The code size for Tinytate is considerably smaller than in our implementation, but Tinytate offers a very limited security level with much smaller parameters that are considered as insecure in today's security systems. Timings on Tmote Sky for the same operation are equal to 4.61s which is more than twice as fast as reported for NanoECC. Also the memory footprint in our version is smaller by 12KB.

Our improvements in pairing calculations are even more impressive in the case of the η_T algorithm. On the Atmega128L we were able to compute it in 2.66s. Other available implementations: NanoECC [44] and TinyPBC [33] are slower with 10.96s and 5.45s respectively. They also use more program space. The biggest difference can be reported on the Tmote Sky platform where our program computes the η_T pairing in 1.71s which is over 3 times faster compared to the 5.25s reported for NanoECC. The reduction in code size is equal to 6.6KB. Even after this reduction, the code consumes 23.66KB, almost half of the available memory on the Tmote Sky node. Currently we are working on different ways to further decrease the memory footprint of our pairing implementations. A success in this direction will remove the last barrier that stands in a way of using practical pairing based mechanisms to secure wireless sensor networks.

7. DISCUSSION AND CONCLUSIONS

There is no doubt that in the future wireless senor networks will be deployed on a large scale in many different parts of the world. Millions of wireless nodes will cooperate to gather crucial data and many of those tiny computers will have a connection with the global network. It is clear that this revolution will put security solutions to a great test.

Researchers in WSN area are very excited about all this pervasive computing and the future "internet of things", but they seem to forget how important the problems of security and privacy are in such an environment. Existing mechanisms for WSNs are still in their early stages and the level of security provided is not satisfactory for many applications.

WSNs are obviously a very challenging environment for applying security services. They differ in many aspects from traditional fixed networks and standard cryptographic solutions cannot be used in this application space. The sole usage of simple and energy efficient symmetric cryptography primitives doesn't solve the security problem at all. On the other hand the whole Public Key Infrastructure which uses asymmetric techniques, digital signatures and certificate authorities seems to be far too complex for a constrained WSN environment. There is a need to develop new security solutions that make use of the latest achievements in cryptography. We believe that the use of pairings and identity based encryption will solve many problems and apply a sufficient level of security to WSNs.

In our work we have presented the state-of-the-art in the implementation of pairing-based cryptography for wireless sensor networks. We have tested our programs on a wide range of different WSN processors and as far as we know all our implementations are the fastest available at the moment. On a system level we have proposed a novel variant of a key exchange protocol which is based on a type-3 pairing and may have some advantages in more demanding applications. We have also investigated the application of a simple non-interactive key exchange protocol that uses a Type-1 pairing. As we have shown the computation of this type of pairing can be performed in a very quick and efficient way, even on the most constrained devices. PBC offers a flexible cryptographic primitive that can be used in many new security protocols and our implementations facilitate the development of such mechanisms.

This study is not only relevant to WSNs but also to different classes of embedded systems where energy and computation power are limited. Examples of those might be Ad-hoc and other distributed networks where security issues are still open problems. Xscale chips are also used in palmtop class devices and our software can be used to secure mobile systems that are build upon this platform. The efficiency of such security mechanisms would be much greater due to the improved capabilities both in terms of available memory and processing power.

A hardware implementation of pairings for WSNs would be several magnitudes quicker and would save a lot of program space, and consume less energy. This would be especially beneficial for very constrained devices with critical amounts of memory. The main drawback of this approach is that a dedicated hardware accelerator would lift the total cost of a mote and would complicate its design. This of course goes against the low-cost/small size philosophy which is a priority for WSN devices. Some work has already been made on this topic [28], [21], but pairings implementation in hardware for wireless sensor networks is still an open research problem that needs to be further investigated.

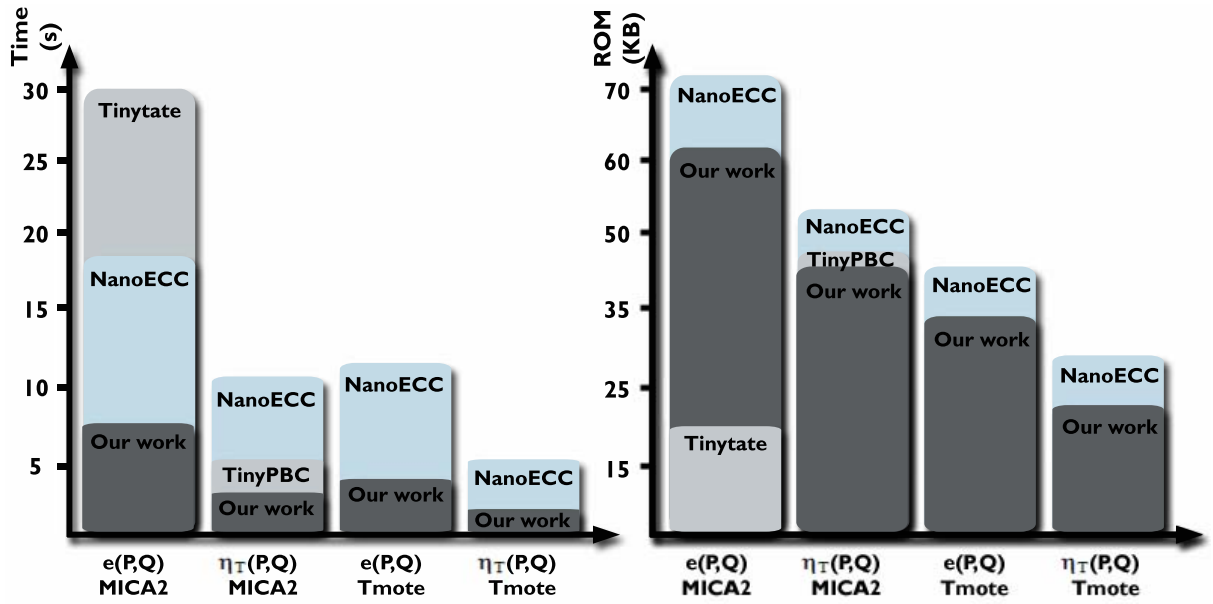

Figure 2: Comparison of different pairing implementations on WSN motes

The extensive study in this paper of PBC implementations in software shows their practicality for use in WSNs. This approach has many advantages like flexibility, fast development time, and low cost. Deploying software-based solutions such as those described here will demonstrate the validity of PBC in the WSN arena, and should motivate the incorporation of hardware support for PBC in future mote designs.

8. REFERENCES

[1] Atmel. ATmega128(L) datasheet, 2006. *http://www.atmel.com*.

[2] P. S. L. M. Barreto, S. Galbraith, C. O'hEigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. *Designs, Codes and Cryptography*, 42:239–271, 2007.

[3] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology – Crypto'2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer-Verlag, 2002.

[4] E.-O. Blaß and M. Zitterbart. Towards Acceptable Public-Key Encryption in Sensor Networks. In *The 2nd Int'l Workshop on Ubiquitous Computing*. ACM SIGMIS, May 2005.

[5] M. Bodrato. Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In C. Carlet and B. Sunar, editors, *WAIFI 2007 proceedings*, volume 4547 of *LNCS*, pages 116–133. Springer, June 2007. *http://bodrato.it/papers/#WAIFI2007*.

[6] S. Chatterjee, P. Sarkar, and R. Barua. Efficient computation of tate pairing in projective coordinate over general characteristic fields. In *Information Security and Cryptology Ű- ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 168Ű–181, 2005.

[7] B. Doyle, S. Bell, A. F. Smeaton, K. McCusker, and N. O'Connor. Security considerations and key negotiation techniques for power constrained sensor networks. *The Computer Journal (Oxford University Press)*, 49(4):443–453, 2006.

[8] R. Dupont and A. Enge. Provably secure non-interactive key distribution based on pairings. *Discrete Appl. Math.*, 154(2):270–276, 2006.

[9] K. Fong, D. Hankerson, J. Lopez, and A. Menezes. Field inversion and point halving revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, 2004.

[10] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. Cryptology ePrint Archive, Report 2006/372, 2006. *http://eprint.iacr.org/2006/372*.

[11] S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *Algorithm Number Theory Symposium – ANTS V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer-Verlag, 2002.

[12] S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165, 2006. *http://eprint.iacr.org/2006/165*.

[13] S. Galbraith and V. Rotger. Easy decision diffie-hellman groups. *LMS Journal of Computation and Mathematics*, 7:201–218, 2004.

[14] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES'04)*, pages 119–132, 2004.

[15] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[16] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS*, 2000.

[17] F. Hess, N. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10), 2006. *http://eprint.iacr.org/2006/110*.

[18] Intel-Corporation. *Intel Xscale Microarchitecture Datasheet*, 2000. *http://www.intel.com*.

[19] A. Joux and R. Lercier. Discrete logarithms in GF(2^{607}) and GF(2^{613}), 2005. *http://perso.univ-rennes1.fr/reynald.lercier/file/nmbrJL05a.html*.

[20] C. Karlof, N. Sastry, and D. Wagner. Tinysec: A link layer security architecture for Wireless Sensor Networks. In *2nd ACM SensSys*, pages 162–175, Nov 2004.

[21] M. Keller, T. Kerins, and W. P. Marnane. FPGA implementation of a GF(2^{4M}) multiplier for use in pairing based cryptosystems. In T. Rissa, S. J. E. Wilton, and P. H. W. Leong, editors, *FPL*, pages 594–597. IEEE, 2005.

[22] T. Kleinjung. Discrete logarithms in GF(p) – 160 digits, 2007. *http://www.nabble.com/Discrete-logarithms-in-GF(p)—-160-digits-td8810595.html*.

[23] D. E. Knuth. *The art of computer programming, volume 2*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[24] A. K. Lenstra. Unbelievable security. Matching AES security using public key systems. In *Advances in Cryptology – Asiacrypt 2001*, volume 2248, pages 67–86. Springer-Verlag, 2001.

[25] S. Lindsey and C. S. Raghavendra. Pegasis: Power-efficient gathering in sensor information systems, 2002.

[26] A. Liu, P. Kampanakis, and P. Ning. Tinyecc: Elliptic Curve Cryptography for sensor networks (ver. 1.0), February 2007. *http://discovery.csc.ncsu.edu/software/TinyECC/*.

[27] D. J. Malan, M. Welsh, and M. D. Smith. A Public-Key Infrastructure for key distribution in TinyOS based on Elliptic Curve Cryptography. In *1st IEEE Intl' Conf. on Sensor and Ad Hoc Communications and Networks (SECON'04)*, 2004.

[28] K. McCusker, N. O'Connor, and D. Diamond. Low-energy finite field arithmetic primitives for implementing security in Wireless Sensor Networks. In *2006 Intl' Conf. on Communications, CircuiTS aND sYstems*, volume III - Computer, Optical and Broadband; Communications; Computational Intelligence, pages 1537–1541, 2006.

[29] A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[30] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on Fundamentals*, E84-A(5):1234–1243, 2001.

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

[32] L. B. Oliveira, R. Dahab, J. Lopez, F. Daguano, and A. A. F. Loureiro. Identity-based encryption for sensor networks. In *5th IEEE International Conference on Pervasive Computing and Communications Workshops PERCOMW '07*, pages 290–294, 2007.

[33] L. B. Oliveira, M. Scott, J. Lopez, and R. Dahab. TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. Cryptology ePrint Archive, Report 2007/482, 2007. *http://eprint.iacr.org/*.

[34] L. Oliviera, D. Aranha, E. Morais, F. Daguano, J. Lopez, and R. Dahab. Tinytate: Computing the tate pairing in resource-constrained sensor nodes. In *6th IEEE Inernational Symposium on Network COmputing and Applications – NCA 2007*, 2007.

[35] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, Sept. 2002. Also appeared in MobiCom'01.

[36] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, 2000.

[37] M. Scott. Computing the Tate pairing. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 2005.

[38] M. Scott. Implementing cryptographic pairings. In *Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 177–196. Springer-Verlag, 2007.

[39] M. Scott. MIRACL – Multiprecision Integer and Rational Arithmetic C/C++ Library, 2007. *http://ftp.computing.dcu.ie/pub/crypto/miracl.zip*.

[40] M. Scott. Optimal Irreducible Polynomials for GF(2^m). Cryptology ePrint Archive, Report 2007/192, 2007. *http://eprint.iacr.org/*.

[41] M. Scott and P. Szczechowiak. Optimizing multiprecision multiplication for public key cryptography. Cryptology ePrint Archive, Report 2007/299, 2007. *http://eprint.iacr.org/2007/299*.

[42] D. Seal. *ARM Architecture Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[43] S. C. Seo, D.-G. Han, and S. Hong. TinyECCK: Efficient Elliptic Curve Cryptography implementation over GF(2^m) on 8-bit MICAz mote. Cryptology ePrint Archive, Report 2008/122, 2008. *http://eprint.iacr.org/*.

[44] P. Szczechowiak, L. Oliviera, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the limits of Elliptic Curve Cryptography in Sensor Networks. In *Wireless Sensor Networks – EWSN 2008*, volume 4913 of *Lecture Notes in Computer Science*, pages 305–320. Springer-Verlag, 2008.

[45] Texas-Instruments. *MSP 430F1611 Datasheet*, 2002. *http://www.ti.com*.

[46] H. Wang, B. Sheng, and Q. Li. Elliptic Curve Cryptography based access control in sensor networks. *International Journal of Security and Networks (IJSN). Special Issue on Security Issues on Sensor Networks*, 1(3/4):127–137, 2006.

[47] R. J. Watro, D. Kong, S. fen Cuti, C. Gardiner, C. Lynn, and P. Kruus. Tinypk: securing sensor networks with public key technology. In *2nd ACM Workshop on Security of ad hoc and Sensor Networks (SASN'04)*, pages 59–64, Washington, DC, October 2004.

[48] S. Zhu, S. Setia, and S. Jajodia. LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *10th ACM conference on Computer and communication security (CCS'03)*, pages 62–72. ACM Press, 2003.