Software Implementation of Pairing-Based Cryptography on Sensor Networks Using the MSP430 Microcontroller

Conrado Porto Lopes Gouvêa and Julio López

Instituto de Computação, Universidade Estadual de Campinas
conrado.gouvea@students.ic.unicamp.br, jlopez@ic.unicamp.br

Abstract. The software implementation of cryptographic schemes for wireless sensor networks poses a challenge due to the limited capabilites of the platform. Nevertheless, its feasibility has been shown in recent papers. In this work we describe a software implementation of pairing-based cryptography and elliptic curve cryptography for the MSP430 microcontroller, which is used in some wireless sensors including the Tmote Sky and TelosB. We have implemented the pairing computation for the MNT and BN curves over prime fields along with the ECDSA scheme. The main result of this work is a platform-specific optimization for the multiplication and reduction routines that leads to a 28% speedup in the field multiplication compared to the best known timings published. This optimization consequently improves the speed of both pairing computation and point multiplication.

Keywords: pairing based cryptography, wireless sensor networks, software implementation.

1 Introduction

Wireless sensor networks (WSN) have been the subject of a lot of research recently due to their vast number of applications. One of the challenges they bring is how to secure their communication against eavesdropping or malicious manipulation. These can be addressed through many cryptographic schemes; but since these nodes are highly constrained environments, these schemes must be implemented with great efficiency.

The advantages of asymmetric over symmetric cryptography for WSNs is well established in the literature. For that reason, we chose to implement two types of asymmetric cryptosystems: pairing-based and elliptic curve cryptography. The security levels being considered are the 64/70-bit, being the most feasible and where most of the work so far has focused; and the 128-bit, which can be expensive but may be necessary in the coming years and has not been well explored for WSNs. The main contributions of this work are a platform-specific optimization to improve the speed of both types of cryptosystems and timings for computations in those two different security levels.

B. Roy and N. Sendrier (Eds.): INDOCRYPT 2009, LNCS 5922, pp. 248–262, 2009.

The remainder of this work is organized as follows. In Section 2 we give an introduction to the MSP430 microcontroller, describing its features and limitations. Subsequently, in Section 3, the fundamental operations of multiplication and reduction are described along with our proposed optimization. The implementation and results of pairing-based cryptography is described in Section 4. In Section 5, the implementation and results of elliptic curve cryptography is detailed. Finally, this paper in concluded in Section 6.

2 The MSP430 Microcontroller

The MSP430 from Texas Instruments is a family of 16-bit microcontrollers mostly known for its low power consumption and it is used in wireless sensors such as the Tmote Sky from Moteiv and the TelosB from Crossbow. It features 12 general purpose registers and a 27 instructions set including one bit only shifts and byte swapping. Memory (bytes and words) can be addressed through four addressing modes: *register direct*, *register indexed* (with an offset word), *register indirect* and *register indirect with post-increment*. Destination operands can be addressed only with register direct and indexed modes.

Each instruction can be represented by up to three words (one for the instruction and two offset words). With only a few exceptions, it is relatively simple to calculate the number of cycles spent in each instruction: one for each word in the instruction, plus one for each memory read and two for each memory write. Short immediate constants (-1, 0, 1, 2, 4 and 8) can be encoded without using offset words with a clever usage of two special registers (for example, zeroing a register the "naive way" – moving 0 to it – takes only one cycle).

Still, there is a critical issue with the instruction set: it lacks both multiply and divide. This is partially addressed with a hardware multiplier present in some of the MSP430 models. It is a memory mapped peripheral that supports four operations: *multiply*, *signed multiply*, *multiply and accumulate* and *signed multiply and accumulate*. In order to use them, it is necessary to write the first operand into one of four specific addresses (MPY, MPYS, MAC, MACS; respectively) according to the operation to be issued. Then, the second operand can be written into another specific address (OP2) and the double precision result will be available with a two cycle delay in two addresses (RESLO, RESHI). The *multiply and accumulate* operations also set the carry flag of the addition into another address (SUMEXT).

An important consequence of the hardware multiplier is that it implies an unusual overhead since the operands must be written to and read from memory. Also, there is no instruction for division, therefore it must be carried out in software which is rather expensive.

When timing the algorithms, we have measured the number of cycles taken by the procedures. Timings in seconds or milliseconds are calculated assuming a 8,000,000 Hz clock; the exact maximum clock varies in each device from the MSP430 family. For that reason, it is recommended to compare running times

by their number of cycles. We have used the MSPGCC compiler version 3.2.3 with the `-O2` optimization flag unless noted otherwise.

3 Multiplication and Reduction

Field multiplication over $\mathbb{F}_p$ sums about 75% of the running time of point multiplication and pairing computation. Consequently, it is crucial to implement it using assembly language since this leads to a speedup greater than two-fold, according to our experiments. Multiplication in $\mathbb{F}_p$ consists of two operations: the plain multiplication of the operands into a double precision number and its subsequent reduction modulo a prime.

3.1 Multiplication

The standard algorithm for multiplication is the Comba method [1], which is a column-wise variant of the row-wise standard schoolbook version that reduces memory accesses. Recently, it has been suggested a variant of the Comba method, the Hybrid method [2], that mixes the row-wise and column-wise techniques. It can be seen as the plain Comba method, with the difference that each "digit" is now stored in multiple machine integers, and the digit-digit multiplication is carried out with the row-wise schoolbook technique. Both methods are illustrated in Figure 1.

The advantage of the Hybrid method is that, in a digit-digit multiplication, all of the integers of the first digit can be stored in registers, reducing memory reads. Consequently, this method is appropriate for platforms with a relatively large number of registers. In [3], the authors present an even more optimized version of the Hybrid method, using *carry-catcher* registers in order to simplify its carry handling. They have also studied its application on many platforms, including the MSP430, where they were able to obtain a 15.4% speed improvement compared to the Comba method.

It appears that the Hybrid method is always superior to the plain Comba method when there are sufficient registers available, but this fails to take into account the characteristics of the platform. Analyzing the running time of the Comba method, it can be concluded that the majority of the time is spent at one repeated step: multiply and accumulate. For each column of the result, it is necessary to compute many products and accumulate them in order to obtain the result of that column and the carries of the next two columns. The importance of the multiply and accumulate step (which we will refer to as "MulAcc") was noted before in [2,4]. However, what has been overlooked so far is the fact that the MulAcc is exactly what is provided by the MAC (Multiply and Accumulate) operation of the MSP430 hardware multiplier.

The MulAcc step is illustrated in Figure 2. It consists of the reading of two integers, one from each operand, followed by their multiplication into a double precision integer, and finally the addition of those two integers to a triple precision accumulator (the third only accumulates the carries of those additions).

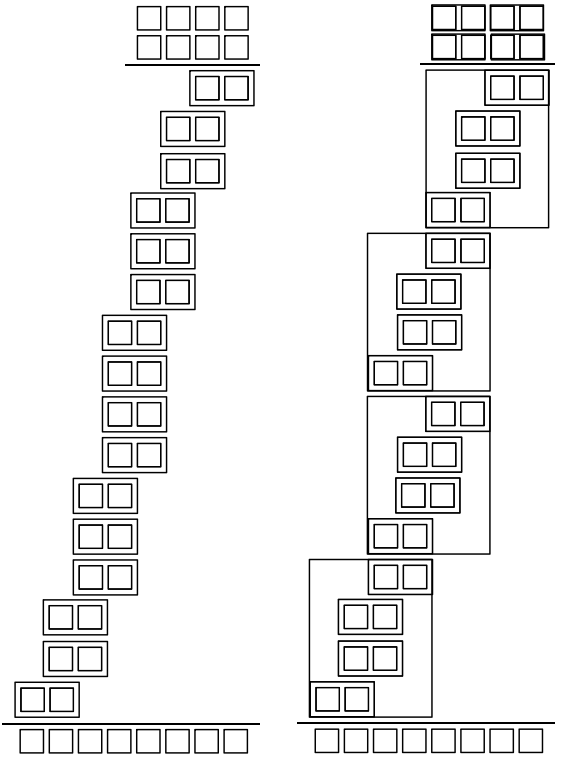

Fig. 1. Comparison of multiplication methods: Comba to the left, Hybrid to the right

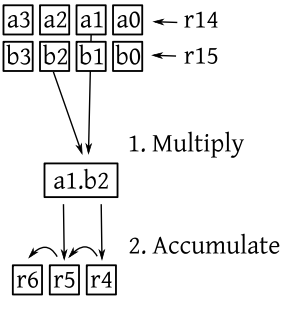

Fig. 2. The MulAcc step, using as example the step for words a1 and b2. The registers `r14` and `r15` hold the pointers to the two 4-word operands

The pseudo-assembly code for the MulAcc step without using MAC is listed in Algorithm 1 and using MAC in Algorithm 2. Compared to Algorithm 1, Algorithm 2 has two less instructions, one less memory read and one less address in extension words, saving four cycles in total. This leads to a great speedup since the MulAcc step is repeated n^2 times with n being the size of the operands in machine integers.

Algorithm 1. Plain MulAcc step

Input: x, the offset address of an integer in the first operand (pointed by `r14`);
y, the offset address of an integer in the second operand (pointed by `r15`)
Ouput: The multiplication of the integers and their accumulation into `r4`, `r5`, `r6`

```
mov x(r14),&__MPY ;move first operand, specify unsigned multiplication
mov y(r15),&__OP2 ;move second operand
add &__RESLO,r4 ;add low part of the result
addc &__RESHI,r5 ;add high part of the result
adc r6 ;add the carry
```

Algorithm 2. MulAcc step using MAC

Input: x, the offset address of an integer in the first operand (pointed by `r14`);
y, the offset address of an integer in the second operand (pointed by `r15`)
Ouput: Multiplication and accumulation into `RESLO`, `RESHI`, `r6`

```
mov x(r14),&__MAC; move first operand; specify multiply and accumulate
mov y(r15),&__OP2 ;move second operand
add &__SUMEXT,r6 ;add the carry
```

The main advantage of using plain Comba with MAC compared to the Hybrid method is that the latter uses all of the 12 available registers, while the former leaves 8 free registers. These can be used as a simple cache for the operands. Additionally, one register can be used to save the address of the SUMEXT in order to add using the *register indirect mode* instead of *register indexed*, saving one more cycle in each MulAcc step (this requires a reordering of the instructions since otherwise the SUMEXT is fetched before the two cycle delay of the hardware multiplier). Table 1 compares the instruction counts of our implementation and those from [3]. It can be readily seen that the greatest savings come from the smaller number of add instructions, since the hardware multiplier does most of the additions by itself. Also, one cycle can be saved in each step due to the linear nature of the access of the first operand, which can be read with the *register indirect with post-increment* addressing mode (`mov @reg+,&label`).

The multiplication timings are detailed in Table 2, where is clear that the Comba multiplier using the MAC optimization is indeed effective, and 9.2% faster than the Hybrid multiplier given in [3]. We have found that using Karatsuba multiplication with a 128-bit Comba multiplier is a little faster than using 256-bit Comba, and it also requires less code space.

Table 1. Comparison of instruction counts of 160-bit multiplication

		Comba MAC		Hybrid in [5]	
Instruction	CPI	Instructions	Cycles	Instructions	Cycles
add @reg,reg	2	99	198		
Other additions				309	709
mov x(reg),&label	6	20	120	45	270
mov reg,x(reg)	4			20	80
mov reg,reg	1			27	27
mov reg,&label	4	89	356	100	400
mov x(reg),reg	3	13	39	45	135
mov @reg+,&label	5	100	500		
mov @reg,&label	5	29	145		
mov @reg,x(reg)	5	20	100		
other			128		167
Totals			1586		1746

Table 2. Timings for multiplication and squaring

Algorithm	Cycles	Time (ms)
160-bit multiplication		
Hybrid in [3]	1,746	0.22
Comba MAC	1,586	0.20
160-bit squaring		
Comba MAC	1,371	0.19
256-bit multiplication		
Hybrid (Karatsuba, 128-bit Comba)	4,025	0.50
Comba MAC (Karatsuba, 128-bit Comba)	3,597	0.45
Comba MAC (256-bit Comba)	3,689	0.46
256-bit squaring		
Comba MAC (Karatsuba, 128-bit Comba)	2,960	0.37

3.2 Reduction

Traditional modular reduction can be an expensive operation because it needs costly divisions. Since the MSP430 has no divide instruction at all, they would need to be computed in software, which would be even more prohibitive. We have selected two algorithms in the literature that do not require divisions: Montgomery reduction [6] and Barrett reduction [7].

Montgomery reduction requires the operands to be transformed into a special Montgomery form. This is often not a problem since we can use the Montgomery

Table 3. Timings for reduction

Algorithm	Cycles	Time (ms)
Modulo 160-bit prime		
Montgomery in [5] (estimated)	2,988	0.37
Montgomery MAC	1,785	0.22
SECG (prime: $2^{160} - 2^{31} - 1$)	342	0.04
Modulo 256-bit prime		
Montgomery	4,761	0.60
Montgomery MAC	3,989	0.50
Barrett	4,773	0.60
NIST (prime: $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$)	709	0.09

form as the "official" representation of all numbers in the cryptographic protocol being used and they would only need to be converted back, for example, to be printed on the screen for human reading. Montgomery reduction also requires a precomputed constant that is dependent of the machine integer size.

The Montgomery reduction algorithm has almost the same structure as the Comba multiplication, with the first operand being the lower part of the double precision number to be reduced and the second operand being the prime modulus. Therefore, one can employ the same MAC optimization to speed up the reduction.

Barrett reduction is slightly more complex and it involves half precision Comba multiplications. Each of these multiplications can also use the MAC optimization. It also requires a precomputed constant which is dependent of the prime modulus.

There also are specific algorithms for reduction when the prime modulus has a special form. For primes of the form $2^k - c$ such as the 160-bit primes from the SECG standard [8] the algorithm is described in [9]. For "NIST primes" [10], the algorithm is described in [11].

The reduction timings are presented in Table 3. The reduction timing from [5] was estimated by subtracting the reported multiplication timing in [3] from the field multiplication timing in [5]. While an exact comparison may be hard to

Table 4. Timings for field multiplication (using Montgomery reduction)

Algorithm	Cycles	Time (ms)
160-bit		
Hybrid in [5]	4,734	0.59
MAC	3,389	0.42
256-bit		
Hybrid	8,855	1.11
MAC	7,604	0.95

make due to this inexact estimate, we notice again that the MAC optimization is very effective. The Barrett reduction was slower than Montgomery reduction, but since we have focused on optimizing Montgomery, we believe its speed can be further improved. As expected, reduction modulo a special form prime is much faster.

Finally, the running times of algorithms for field multiplication – multiplication followed by reduction – are given in Table 4. Compared to [5], field multiplication using MAC is about 28% faster.

4 Identity Based Cryptography Using Pairings

It has been shown recently that identity-based cryptography using bilinear pairings is very appropriate in the wireless sensor network scenario [12]. There are many identity-based cryptographic schemes, but the most useful in this context probably is the non-interactive key agreement scheme [13,14,15] that allows two parties to compute a mutual key without interaction in order to bootstrap a secure channel using symmetric encryption, and will be described next.

Let $e : G_1 \times G_2 \rightarrow G_{\mathrm{T}}$ be a bilinear pairing with G_1 and G_2 being additive groups and G_{T} a multiplicative group, all of them with a prime order r. Let $H_1 : \{0,1\}^* \rightarrow G_1$ and $H_2 : \{0,1\}^* \rightarrow G_2$ be two hash functions. The master key generation is done by the key generation center by choosing a random $s \in \{1, ..., r-1\}$. The private key distribution is done before the deployment of the sensors by assigning a sensor A the identity ID_{A} and private keys $S_{1\mathrm{A}} = sH_1(\mathrm{ID_A})$ and $S_{2\mathrm{A}} = sH_2(\mathrm{ID_A})$.

Now, suppose sensors A and B wish to compute a shared key. If G_1 and G_2 were the same group and the pairing was symmetric, then the two hash functions would be the same and the two private keys of each node would be equal. Therefore, A could compute $e(S_{1\mathrm{A}}, H_1(\mathrm{ID_B}))$ and B could compute $e(H_1(\mathrm{ID_A}), S_{1\mathrm{B}})$. Due to the bilinearity and symmetry, we have

$$\begin{aligned} e(S_{1\mathrm{A}}, H_1(\mathrm{ID_B})) &= e(sH_1(\mathrm{ID_A}), H_1(\mathrm{ID_B})) \\ &= e(H_1(\mathrm{ID_A}), sH_1(\mathrm{ID_B})) \\ &= e(H_1(\mathrm{ID_A}), S_{1\mathrm{B}}) \\ &= e(S_{1\mathrm{B}}, H_1(\mathrm{ID_A}))\ . \end{aligned}$$

Then both parties can generate the same value, which can be used to derive a shared key. In our case, though, the pairing is asymmetric since the elliptic curves used are ordinary. Therefore, we need two private keys for each sensor, the hash functions are different, and the last step in the equation is not valid. Still, we have two useful equations which can be easily verified: $e(S_{1\mathrm{A}}, H_2(\mathrm{ID_B})) = e(H_1(\mathrm{ID_A}), S_{2\mathrm{B}})$ and $e(H_1(\mathrm{ID_B}), S_{2\mathrm{A}}) = e(S_{1\mathrm{B}}, H_2(\mathrm{ID_A}))$. In [14], it is suggested

that each party should multiply their sides of those two equations in order to compute the shared key, but this requires two pairing computations. In [5] it is suggested that the sensors could agree on which equation they should use with a little amount of communication. Instead, there is a simpler fix that maintains the non-interactive aspect of the protocol. It can be defined that the sensor with the smaller ID in lexicographical order should use its first private key in the first pairing parameter and the other its second private key in the second pairing parameter, therefore choosing one of the equations without any interaction.

4.1 MNT Curve over a 160-Bit Field

For 160-bit fields, we have implemented two security levels. To allow comparisons, the first one is the same described in [5] which uses a MNT curve of embedding degree 4. These parameters where chosen in order to provide minimum acceptable security; the 640-bit extension field used gives approximately 64 bits of security [16]. The authors chose the Tate pairing instead of the faster Ate pairing since hashing a identity to a point in G_2 is simpler in the Tate pairing. The Miller loop is implemented using the sliding window technique with $w = 3$. The second level of security chosen follows a similar implementation but using a MNT curve with embedding degree 6. This results in a 960-bit extension field that provides approximately 70 bits of security [17].

The respective finite field operation and pairing computation timings are detailed in Table 5, which shows that the MAC optimization leads to a 20.2% speedup in the 64-bit level. It is important to remark that in [5] the authors chose to compile their code with optimization turned off; the reason given is that the difference in speed obtained by using different compilers is very significant when using optimization and that would make any comparisons harder.

Table 5. Timings for field operations and pairing computations on MNT curves

Algorithm	Optimization	Cycles	Time (ms)
Field operations			
Multiplication		3,389	0.42
Squaring		3,172	0.40
Inversion		187,575	23.45
MNT curve, $k = 4$			
Tate [5]	Off	37,739,040	4,717
Our Tate (MAC)	Off	30,125,088	3,766
Our Tate (MAC)	On	26,553,690	3,319
MNT curve, $k = 6$			
Our Tate (MAC)	Off	51,199,102	6,400
Our Tate (MAC)	On	40,869,215	5,109

Still, we feel that providing the timings for the optimized versions would lead to more interesting comparisons.

4.2 BN Curve over a 256-Bit Field

For the 128 bits security level, the Barreto-Naehrig family of curves [18] was chosen. They have an embedding degree of 12 and provide a sextic twist that allows the doubling and adding of Miller's algorithm to be performed on the curve over $\mathbb{F}_{p^2}$ instead of the costly $\mathbb{F}_{p^{12}}$. The curve chosen is the one generated by the x value of $-0x4080000000000001$ suggested in [19]. Regarding the BN formulas, one can find in the literature different values for $p(x)$: the original paper [18] uses $p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$ but some other papers [19,20] use $p(x) = 36x^4 - 36x^3 + 24x^2 - 6x + 1$, which gives the same value when using x with inverted sign. We use the original version.

The pairings chosen were the Optimal Ate [21], R-ate [22] and Xate [19]; all of them optimal pairings as defined in [21]. They provide optimal speed by truncating the Miller loop by a quarter. We follow the approach detailed in [20] but using the final exponentiation optimization from [23]. Since the Miller loop runs through the bits of $6x + 2$ (or x in Xate), which has low Hamming weight, the sliding window technique is not appropriate and was not used.

We present the timings for the finite field operations and pairing computations in Table 6. The pairing computation is much more expensive than in the MNT curve, and probably unacceptable for the wireless sensor scenario. As noted in [24], it is important to keep in mind that the pairing computation scales more-or-less like RSA rather than like elliptic curve cryptography. It is also worth noticing that the three kind of pairings give almost the same speed, with the Xate pairing being a little faster. We describe the Xate pairing for BN curves in Algorithm 3.

The ROM and RAM requirements of the pairing computation program are listed in Table 7. To put them in perspective, we note that popular sensors have such as Tmote Sky and TelosB have 48KB of ROM and 10K of RAM. The code size is still large; though it is only possible to determine its feasibility by analyzing specific applications. The amount of RAM allocated is probably tolerable, since most of it is allocated from the stack and freed after the computation.

5 Elliptic Curve Cryptography

While identity based schemes built with pairings seem ideal for the wireless sensor scenario, they still are expensive, mainly in the higher 128-bit level of security. For that reason, we have also implemented the cheaper elliptic curve cryptography in order to allow comparison with pairing-based cryptography. To illustrate a concrete use, the ECDSA (Elliptic Curve Digital Signature Algorithm) [10]

Algorithm 3. Xate pairing for BN curves

Input: $x \in \mathbb{Z}$ (the BN parameter), $Q \in E'(\mathbb{F}_{p^2})$, $P \in E(\mathbb{F}_p)$
Ouput: $\zeta(Q,P) \in \mathbb{F}_{p^{12}}$
1: $v, xQ \Leftarrow f_{|x|,Q}(P)$ $\{f_{r,Q}$ if the Miller function, it also computes $rQ\}$
2: **if** $x > 0$ **then**
3: $\quad v \Leftarrow 1/v$
4: $\quad xQ = -xQ$
5: **end if**
6: $v \Leftarrow v^{1+p+p^3+p^{10}}$
7: $v, A \Leftarrow g_{xQ,pxQ}(P)$ $\{g_{P,Q}$ is the line function from the Miller function, it also computes $P+Q$ $\}$
8: $v, B \Leftarrow g_{p^3xQ,p^{10}xQ}(P)$
9: $v, C \Leftarrow g_{A,B}(P)$
10: **return** $v^{(p^{12}-1)/r}$

Table 6. Timings for field operations and pairing computations on the BN curve

Algorithm	Cycles	Time (ms)
Field operations		
Multiplication	7,569	0.95
Squaring	6,952	0.87
Inversion	380,254	47.53
Pairings		
Optimal Ate	117,597,798	14,700
R-ate	117,514,219	14,689
Xate	116,130,546	14,516

Table 7. ROM and maximum allocated RAM size for pairing programs

Version	ROM (KB)	RAM (KB)
BN 256 bits, Karatsuba w/ Comba 128	32.3	4.7
BN 256 bits, Comba 256	36.2	4.7
MNT 160 bits, Comba 160	28.9	2.3
MNT 160 bits, Comba 160 in [5]	34.9	3.4

was chosen for its popularity and wide standardization. However, it is important to notice that elliptic curve cryptography still requires the expensive public key authentication which is outside the scope of this work.

The ECDSA is composed by key generation, signature generation and verification. The key and signature generation require a fixed point multiplication that is their most expensive operation. In our implementation, we have used the Comb algorithm with window size 4 [11] which requires the

precomputation of 15 elliptic curve points. For the signature verification, we have used the interleaving algorithm with NAF [11] of width 5 and 4 for the fixed and random points, respectively.

At the 80-bit level of security, the secg160r1 [8] curve was chosen which allows fast reduction [9] due to its special form modulus. This curve has -3 as its b parameter to enable a known optimization in the point doubling. At the 128-bit level of security, the P-256 curve [10] was chosen which also provides fast reduction [11] due to its special form modulus ("NIST prime"). This curve also has -3 as its b parameter.

We present the timings for the finite field operations and point multiplication in Table 8 and the ECDSA timings in Table 9. The timings results of our implementation are faster than [25], but they do acknowledge that their work leaves room for much optimization. Also notice that the 5NAF is not adequate since it is just a little faster than 4NAF but requires double storage space. The Montgomery ladder method [26], while secure against side-channel attacks (timing and power analysis), is 40–50% slower than 4NAF.

The ROM and RAM requirements for the ECDSA program are listed in Table 10. The ROM sizes are about 5% smaller than the pairing-based cryptography, and seem to be acceptable, specially in the 80-bit level of security. The RAM requirements are also realistic since most of it is freed after the computation.

Table 8. Timings for field operations and point multiplication for the given curves

	secg160r1		P-256	
Algorithm	Cycles	Time (ms)	Cycles	Time (ms)
Field operations				
Multiplication	1,952	0.24	4,327	0.54
Squaring	1,734	0.22	3,679	0.46
Inversion	187,575	19.27	292,170	36.52
Random point multiplication				
4NAF	4,417,661	0.552	13,372,271	1.672
5NAF	4,433,104	0.554	13,188,903	1.649
Montgomery ladder	6.319,383	0.790	20,476,234	2.560
Unknown from [25]		0.800		
Fixed point multiplication				
Comb, $w = 4$	1,831,063	0.229	5,688,793	0.711
Comb, $w = 4$ in [27]		0.720		
Sliding window, $w = 4$ in [25]		0.720		
Simultaneous point mult.				
Interleaved	5,204,544	0.651	15,784,176	1.973

Table 9. Timings for ECDSA

	secg160r1		P-256	
Algorithm	Cycles	Time (s)	Cycles	Time (s)
Key Generation	1,849,903	0.231	5,682,433	0.710
Sign	2,166,906	0.270	5,969,593	0.746
Verify	5,488,568	0.686	16,139,555	2.017

Table 10. ROM and maximum allocated RAM size for elliptic curve programs

Version	ROM (KB)	RAM (KB)
256 bits, Karatsuba w/ Comba 128	25.7	3.5
256 bits, Comba 256	29.5	3.5
160 bits, Comba 160	23.5	2.5
160 bits, Comba 160 in [27]	31.3	2.9

6 Conclusion

Implementing efficient cryptographic schemes on wireless sensor networks is a difficult task, but feasible. It is important to analyze every feature offered by the platform in order to get the best results, as can be seen with the simple but effective optimization using the MAC operation from the hardware multiplier of the MSP430. Still, there is plenty of work to be done. As our implementation has shown, there is a steep price to be paid in the 128-bit level of security pairing computation (14.5 seconds). Some relevant future work that we would suggest is to provide a fast implementation of identity based cryptography in other security levels and implement in software the recently proposed method to speed up finite field arithmetic for BN curves [28].

References

1. Comba, P.: Exponentiation cryptosystems on the IBM PC. IBM Systems Journal 29(4), 526–538 (1990)
2. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.: Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 925–943. Springer, Heidelberg (2004)
3. Scott, M., Szczechowiak, P.: Optimizing multiprecision multiplication for public key cryptography. Cryptology ePrint Archive, Report 2007/299 (2007), `http://eprint.iacr.org/`
4. Großschädl, J.: Instruction Set Extension for Long Integer Modulo Arithmetic on RISC-Based Smart Cards. In: Symposium on Computer Architecture and High Performance Computing, pp. 13–19 (2002)

5. Szczechowiak, P., Kargl, A., Scott, M., Collier, M.: On the application of pairing based cryptography to wireless sensor networks. In: Proceedings of the second ACM conference on Wireless network security, pp. 1–12. ACM, New York (2009)
6. Montgomery, P.: Modular multiplication without trial division. Mathematics of computation 44(170), 519–521 (1985)
7. Barrett, P.: Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 311–323. Springer, Heidelberg (1987)
8. Certicom Research: SEC 2: Recommended Elliptic Curve Domain Parameters (2006), http://www.secg.org/
9. Menezes, A., Van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
10. National Institute of Standards and Technology: FIPS 186-3: Digital Signature Standard (DSS) (2009), http://www.itl.nist.gov
11. Hankerson, D., Vanstone, S., Menezes, A.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)
12. Oliveira, L., Aranha, D., Morais, E., Daguano, F., Lopez, J., Dahab, R.: TinyTate: computing the tate pairing in resource-constrained sensor nodes. In: Sixth IEEE International Symposium on Network Computing and Applications, NCA 2007, pp. 318–323 (2007)
13. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan (2000)
14. Dupont, R., Enge, A.: Provably secure non-interactive key distribution based on pairings. Discrete Applied Mathematics 154(2), 270–276 (2006)
15. Oliveira, L., Scott, M., Lopez, J., Dahab, R.: TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. In: 5th International Conference on Networked Sensing Systems, INSS, pp. 173–180 (2008)
16. Lenstra, A.K.: Key Lengths. In: Handbook of Information Security. John Wiley & Sons, Chichester (2004)
17. Lenstra, A., Verheul, E.: Selecting Cryptographic Key Sizes. Journal of Cryptology 14(4), 255–293 (2001)
18. Barreto, P., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
19. Nogami, Y., Akane, M., Sakemi, Y., Kato, H., Morikawa, Y.: Integer Variable χ-Based Ate Pairing. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209, pp. 178–191. Springer, Heidelberg (2008)
20. Devegili, A., Scott, M., Dahab, R.: Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 197–207. Springer, Heidelberg (2007)
21. Vercauteren, F.: Optimal pairings. Cryptology ePrint Archive, Report 2008/096 (2008), http://eprint.iacr.org/
22. Lee, E., Lee, H.S., Park, C.M.: Efficient and generalized pairing computation on abelian varieties. Cryptology ePrint Archive, Report 2008/040 (2008), http://eprint.iacr.org/
23. Scott, M., Benger, N., Charlemagne, M., Perez, L.J.D., Kachisa, E.J.: On the final exponentiation for calculating pairings on ordinary elliptic curves. Cryptology ePrint Archive, Report 2008/490 (2008), http://eprint.iacr.org/
24. Galbraith, S., Paterson, K., Smart, N.: Pairings for cryptographers. Discrete Applied Mathematics 156(16), 3113–3121 (2008)

25. Wang, H., Li, Q.: Efficient Implementation of Public Key Cryptosystems on Mote Sensors (Short Paper). In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 519–528. Springer, Heidelberg (2006)
26. Montgomery, P.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation 48(177), 243–264 (1987)
27. Szczechowiak, P., Oliveira, L., Scott, M., Collier, M., Dahab, R.: NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In: Verdone, R. (ed.) EWSN 2008. LNCS, vol. 4913, pp. 305–320. Springer, Heidelberg (2008)
28. Fan, J., Vercauteren, F., Verbauwhede, I.: Faster Fp-arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 240–253. Springer, Heidelberg (2009)