Ultra Low-Power implementation of ECC on the ARM Cortex-M0+

Ruan de Clercq, Leif Uhsadel, Anthony Van Herrewege, Ingrid Verbauwhede K.U. Leuven,
Department of Electrical Engineering - ESAT/COSIC and iMinds
Kasteelpark Arenberg 10, 3001 Heverlee-Leuven,
Leuven, Belgium
mail: *firstname.lastname@esat.kuleuven.be*

Abstract

In this work, elliptic curve cryptography (ECC) is used to make an efficient implementation of a public-key cryptography algorithm on the ARM Cortex-M0+. The goal of this implementation is to make not only a fast, but also a very low-power software implementation. To aid in the elliptic curve parameter selection, the energy consumption of different instructions on the ARM Cortex-M0+ was measured and it was found that there is a variation of up to 22.5% between different instructions. The instruction set architecture (ISA) and energy measurements were used to make a simulation of both a binary curve and a prime curve implementation, and the former was found to have a slightly faster execution time with a lower power consumption. Binary curve arithmetic use instructions which requires less energy than prime curve arithmetic on the target platform. A new field multiplication algorithm is proposed, called `López-Dahab with fixed registers`, which is an optimization of the `López-Dahab` (`LD`) algorithm. The proposed algorithm has a performance improvement of 15% over the `LD with rotating registers` algorithm (which is the current fastest optimization of the `LD` algorithm). A software implementation that uses the proposed algorithm was made in C and assembly, and on average our implementation of a random point multiplication requires 34.16 µJ, whereas our fixed point multiplication requires 20.63 µJ. The energy consumption of our implementation beats all known software implementations on embedded platforms, of a point multiplication, on the same equivalent security level by a factor of 7.4.

I. Introduction

A typical application for public-key cryptography in the ultra low-power domain is for Wireless Sensor Network (WSN). A WSN is an ad-hoc wireless network that consists of a number of nodes and one or more base stations. WSNs require security, because they communicate through an insecure communication medium and they often operate unattended. As these devices are made to be economically viable, they have a limited amount of energy, computation power, memory and communication abilities. A node's lifetime is also directly influenced by the amount of energy that it uses to perform computations and is therefore also directly influenced by the efficiency of its algorithms.

Due to their high computational requirements, RSA [1] and DSA [2] are considered to be impractical for use in WSNs, where the devices are very constrained in processing power and energy. ECC [3], [4] is an attractive alternative due to its low computational and memory requirements, and is particularly useful in hybrid cryptosystems where PKC is used for key exchange, and symmetric cryptography is used for the efficient encryption of data. Digital signatures are also useful for WSNs, as they can guarantee the authenticity, and integrity of the data. To perform a key exchange the Elliptic Curve Diffie-Helman exchange (ECDH) can be used, and for generating digital signatures the Elliptic Curve Digital Signature Algorithm (ECDSA) could be used.

The ARM Cortex-M0+ [5] is a low cost, ultra low-power microcontroller that provides great performance due to its 32-bit architecture, and it features a small but powerful instruction set. The authors are convinced that the chosen platform is appropriate for WSNs not only because of its specs, but also as first integrations of this unit are already announced [6]. As this processor has only been available since 2012, we do not know of any other PKC implementations optimized for this architecture. By being the first to make an implementation on this architecture we are setting a benchmark to which the performance of future implementations can be measured.

We now present to you the new state of the art in low-power software implementations of ECC on the ARM Cortex-M0+. Efficient long number arithmetic will be performed on the architecture using assembly optimizations for the processor's instruction set. The results will be compared to the existing solutions in the ultra low-power domain, as well as to a standard library which have been compiled on ARM.

The rest of the paper is organized as follows. First, we will discuss related work in the low-power domain, followed by a discussion on the methods that were used to perform the parameter, and algorithmic selection for our implementation. Next, we describe our results, and compare it with the results from implementations found in literature and in software libraries. Subsequent, we discuss some ideas for future work, and finally we provide a general conclusion.

II. Related Work

Here we will discuss the related work of low-power software implementations of ECC. There is an evolution of algorithms and hardware, and therefore the overview follows a chronological order, with a focus on the `López-Dahab (LD)` [7] field

multiplication method, as our implementation is based on this. The LD method and window parameter w will be discussed in more detail later. A number of low-power implementations exist in the literature; however, in the past a lot of the focus has gone towards software implementations on existing WSNs like the 8-bit MICA2 and MICAz (which both contain the ATMega128L) and the 16-bit TelosB (which contains the MSP430). Only a small number of implementations were found in the literature for ARM microcontrollers like the IMote2 (which contains the ARMv5TE based PXA271), and the ARM7TDMI.

Szczechowiak et al. [8] made binary curve, and prime curve implementations for the Tmote Sky and the MICA2, based on the MIRACL library. Multiplication on prime curves use Hybrid multiplication [9], and multiplication on binary curves use the Karatsuba-Ofman multiplication algorithm. For binary fields they used binary Koblitz curves as no expensive doubling operations are required. For their fixed point multiplication in prime fields they did some pre-computation with the Comb method and $w = 4$. Their point multiplications in prime fields was found to be faster than in binary.

Kargl et al. made software implementations on the ATMega128L for prime, and binary fields [10]. For multiplication in the binary field they use LD with $w = 4$, and a Montgomery-ladder algorithm which provides a constant execution time for point multiplication.

B. Oliveira et al. [11] made an implementation for the PXA27x on an underlying binary field of order 2^{271}. They made an optimization of the LD algorithm, called the `LD with rotating registers` method. They also performed immediate reduction of the upper half of the words instead of writing them to memory for later reduction. Assembly optimizations were used for field arithmetic.

P. Szczechowiak et al. [12] made an implementation in 2^{271} that uses the LD with $w = 8$, and 2-bit scanning. Two pointers are used to access the appropriate bytes in memory, thereby avoiding a multi-precision shift of the partial product vector.

Aranha et al. [13] made an optimization to the LD algorithm, called `LD with rotating registers`, where the memory accesses of intermediate values are reduced by making use of a rotating register scheme. In their implementation on the ATMega128L, they interleave multiplication with the reduction operation, and modular squaring is done with the table-based method interleaved with the reduction step so that the upper half of the words which are produced by squaring doesn't need to be written to memory.

S. Erdem [14] made several binary curve implementations for the ARM7TDMI using the operand-scanning method combined with LD with $w = 4$.

Gouvea et al. [15] made implementations on prime curves, binary curves, and binary Koblitz curves for MSP430 microcontrollers. Comba [16] multiplication is used for 160-bit prime curves, and Karatsuba-Ofman [17] multiplication is used for 256-bit prime curves. For binary fields they use LD multiplication for the 163-bit underlying field, and Karatsuba-Ofman with LD for the 283-bit underlying field.

III. Methods

In this section, we present you with the methods that were used to perform the parameter and algorithmic selection that is necessary to make an efficient and low-power ECC implementation. First, we will discuss the model that was used to make a curve selection. Next, we will discuss some of the algorithmic choices that was made.

A. Matching a curve to the architecture

In order to make an efficient and low-power implementation it is necessary to select the appropriate curve for the architecture of the target platform. A model was made to determine the instruction usage, cycle count, and energy usage of a specific curve. For the model we considered only Binary Koblitz, and prime curves. Efficient algorithms and coordinate systems were selected to perform a point multiplication. The core of this model consisted of an analysis of the instructions required for performing a field multiplication algorithm, as this is most dominant routine in terms of execution time in an ECC. From this we estimated the execution times for performing a point multiplication, and we came to two conclusions: (1) Binary Koblitz curves will lead to a slightly faster implementation (2) Binary curves require less power than prime curves, due to the binary curve arithmetic using many XOR and shift instructions, whereas prime curves require instructions which requires less energy (shift and XOR vs multiply and ADD) on the target platform (See section IV-A).

B. Field arithmetic algorithms

Here we will discuss some of the different field arithmetic algorithms that were used during analysis and implementation.

1) Multiplication: Consider two binary polynomials $x(z)$ and $y(z)$ of degree at most $m - 1$. The output of the field multiplication function should produce the result of the polynomial multiplication $x(z) \cdot y(z)$.

The López-Dahab (LD) field multiplication algorithm is a windowed multiplication algorithm for $\mathbb{F}_{2^m}$. Its goal is to reduce the number of multi-precision shift operations by scanning the input parameter x with w bits at a time and performing a table lookup, thereby reducing the number of outer loop iterations to only $\lceil W/w \rceil$, where W is the word size of the processor. The

lookup table is computed with $T(u) \leftarrow u(z) \cdot y(z)$ for all polynomials $u(z)$ of degree lower than w. The number of words required to store the lookup table is given by:

$$\begin{cases} 2^w(n+1) & \text{, if degree}(y) > nW - (w-1), \\ 2^w(n) & \text{, if degree}(y) \leq nW - (w-1), \end{cases} \tag{1}$$

where n is the number of words needed for the field parameter. This means that if the degree of the most significant word of y is smaller or equal to $W-(n-1)$, then the lookup table will fit into $2^w n$ words. This is due to the fact that while generating the lookup table, y gets shifted by $w-1$ which causes the polynomial to overflow into the next word.

We propose a new optimization to the `(LD)` field multiplication algorithm, and call it the `López-Dahab with fixed registers` method. This algorithm aims to reduce the number of memory operations by keeping as many words of the internal state vector as possible inside registers. The most frequently used words are stored inside fixed register positions, and the least frequently used words are stored inside memory. On the target platform it is feasible to store a maximum of nine words inside registers.

Algorithm 1 shows the `LD with fixed registers` for $n = 8$, and a register count of $n+1$.The vector v denotes the internal state vector of $2n$ words which contains the intermediate results, which are stored inside memory, as well as fixed register positions. The $n+1$ most frequently used words are stored inside registers (r), and the remaining $n-1$ words are stored inside memory (m). It was observed that $v[3 \cdots 12]$ are the most frequently used $n+1$ elements and are therefore stored inside the registers. $v[0 \cdots 2]$ and $v[13 \cdots 15]$ are the least frequently used $n-1$ elements inside v and are therefore stored inside memory.

Fig. 1 provides a visual representation of the algorithm. All the light colored squares represents words which are stored in memory, and all the dark colored squares represents words which are stored in registers. First the lookup table (indicated with LUT) is computed from the input parameter x. Each cell inside the LUT represents 8 words stored in memory. The vector C contains the partial products of the multiplication, and consists of words stored inside memory, as well as in registers. The vector y is split into sections of w bits, and these sections are used as an index into the LUT. The index is used to read a cell in the LUT, and add it to C. As each cell contains 8 words, 8 words are read from the LUT, and then added to C. The lookup and add process is repeated 8 times, each time offset by one more word. After the eighth lookup, C is left shifted by 4 bits. This is repeated 8 times, but in the final iteration the shift is not required.

In order to reduce the number of memory operations the field multiplication algorithm can be interleaved with the reduction algorithm.

Algorithm 1 `López-Dahab with fixed registers` multiplication in $\mathbb{F}_{2^m}$ for $n = 8$.

Input: $x(z) = x[0 \cdots n-1], y(z) = y[0 \cdots n-1]$
Output: $v(z) = v[0 \cdots 2n-1] = x(z)y(z)$
Note: v denotes the internal state vector composed of $n-1$ memory addresses and $n+1$ registers. $v \leftarrow (m[0], m[1], m[2], r_0, r_1, \cdots, r_n, m[3], m[4], m[5], m[6])$.

```
1:  Compute T(u) ← u(z)y(z) for all polynomials u(z) of degree lower than w
2:  v[0 ··· 2n − 1] ← 0
3:  for j ← ⌈W/w⌉ − 1 downto 0 do
4:      for k ← 0 to n − 1 do
5:          u = (x[k] ≫ j · W) & 0xF
6:          for l ← 0 to n − 1 do
7:              v[l + k] ← v[l + k] ⊕ T[u][l]
8:          end for
9:      end for
10:     if (j ≠ 0) then
11:         v(z) = v(z) · z^w
12:     end if
13: end for
14: return v
```

2) Reduction: Since the curve we are using has a sparse reduction polynomial, the reduction can be efficiently computed one word at a time.

3) Inversion: Inversion is done by means of the Extended Euclidean Algorithm [18] for polynomials. This algorithm makes use of two multi-precision internal state variables (u and v). One expensive operation that is called for in the algorithm is swapping u with v. The swap operation can be avoided by writing two separate segments of code in which both segments perform the same operations, but where the names of the variables are interchanged. This eliminates a large number of expensive memory operations. Another optimization is to use two variables to store the index of the most significant non-zero word of

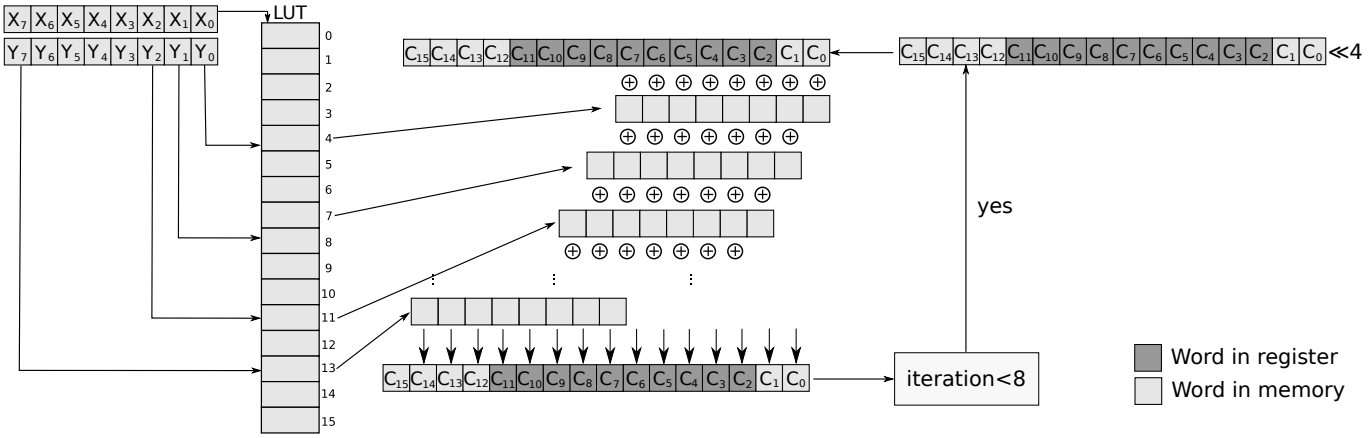

Fig. 1. The proposed `LD with fixed registers` algorithm in $\mathbb{F}_{2^m}$ for $n = 8$. The lookup table LUT is generated from the input scalar x. The main loop is executed 8 times for $w = 4$. The input parameter y is split up into sections of w bits which are used as an index for the LUT.

u and v. This allows for performing a fast calculation of the degree of the polynomial and a reduced number of memory operations in the variable field shift function.

4) Squaring: Squaring is done by means of a 16-bit lookup table with 256 entries, requiring 4 kB. To reduce redundant memory operations, modular reduction is interleaved with the squaring functions. The lower half of the output of the squaring operation is kept inside the registers and the upper half is expanded and then immediately reduced. The upper half of the elements are therefore not required to be stored first and reduced later.

C. Analysis of multiplication algorithms

As field multiplication is the most dominant routine in terms of execution time in an ECC, three field multiplication methods were analyzed to determine the best match for the target platform. For all three methods a window size of $w = 4$ is used, where a single precomputation table of $16n$ words (4 kB) is required. This is valid under the assumption that the scalar y is short.

For both the analysis of the `LD with rotating registers`, and the `LD with fixed registers` methods, we assume that $n + 1$ registers are available for storing the partial products.

TABLE I
ESTIMATED CYCLE REQUIREMENTS FOR FIELD MULTIPLICATION IN $\mathbb{F}_{2^{233}}$.

Method	Read	Write	XOR
A	$16n^2 + 23n$	$8n^2 + 30n$	$8n^2 + 30n - 7$
B	$8n^2 + 39n - 8$	$46n$	$8n^2 + 38n - 7$
C	$8n^2 + 24n + 1$	$31n + 1$	$8n^2 + 30n - 7$

Method A: `LD`
Method B: `LD with rotating registers`
Method C: `LD with fixed registers`
The number of shift operations remain constant at $42n - 21$ for all three methods

TABLE II
ESTIMATED CYCLE REQUIREMENTS FOR FIELD MULTIPLICATION IN $\mathbb{F}_{2^{233}}$.

Method	Read	Write	XOR	Shift	Total*
A	1208	752	745	315	4980
B	816	368	809	315	3492
C	689	233	745	315	2968

Method A: `LD`
Method B: `LD with rotating registers`
Method C: `LD with fixed registers`
The total number of shift operations is constant at $42n - 21$ for all three methods
* Memory operations are assumed to require two cycles per operation

The total number of operations and cycle estimates are shown in Table I and Table II respectively. The cycle estimate assumes that a memory operation will take 2 cycles and all other operations take only 1 cycle to complete. When comparing

the `LD` method to the `LD with rotating registers` method, we see a drastic reduction in the number of memory operations due to the implementation of the rotating register scheme, which minimizes the storing of intermediate values in memory. When comparing the `LD with fixed registers` method to the `LD with rotating registers`, we see a further reduction of memory accesses due to the more efficient usage of registers. The `LD with fixed registers` has a performance increase of 15% over the `LD with rotating registers` method, and a performance increase of 40% over the standard `LD` method.

As we would like to make an implementation that will have the fastest possible execution time we made an implementation with the `LD with fixed registers` algorithm, which has the lowest estimated cost of only 2968 cycles.

D. Point multiplication

Point multiplication is the operation of multiplying a scalar k with a point P on an elliptic curve, and it is responsible for the majority of the execution time of an ECC system. It is defined by the repeated addition of the point P with itself, $k-1$ times:

$$kP = \underbrace{P + P + \cdots + P}_{k-1 \text{ additions}}.$$

Point multiplication can be done by multiplying a scalar with either a fixed point, or with a random point. For the fixed point multiplication, the point can be seen as a constant, and therefore some precomputations can be done on this point in order to speed up the multiplication.

IV. Results

This section will be used to present our key results. First, we will discuss the measurements setup, and the results from our measurements. Next, we will present two implementations, and compare them to the state of the art low-power implementations implementations found in literature, as well as in software libraries

A. Measurements setup and results

In order to determine the energy usage of different instructions as well as cryptographic software implementations, a system was designed to measure the power consumption of the target platform.

The power consumption for a number of different instructions were measured in order to investigate the effect of different field arithmetic algorithms on the overall power consumption. Table III shows the results of energy measurements for instructions which are relevant to prime and binary field arithmetic. A variation in energy consumption of up to 22.5% was observed between different instructions. The ADD instruction was found to be the most energy hungry, requiring 6.9% more energy than any other measured instruction. This is important for the choice of the underlying field because binary field arithmetic require a large amount of shift (LSL and LSR) and XOR instructions, whereas on prime field arithmetic require a large amount of MUL and ADD instructions.

TABLE III
The energy used per cycle for different instructions. The clock frequency is 48MHz.

Instruction	Energy [pJ]
LDR	10.98
LSR	12.05
MUL	12.14
LSL	12.21
XOR	12.43
ADD	13.45

B. Comparison with other libraries

Table IV and Table V shows the proposed implementation compared with low power implementations found in literature, as well as in software libraries. In the cases where the energy consumption were not provided in the author's results, the values are estimated from the typical energy consumption values found in [19], [20]. Both the ARM7TDMI, and the PXA271 are more powerful platforms than the ARMv6-M based ARM Cortex-M0+ because they both have larger instruction sets; however, the ARM Cortex-M0+ uses less energy than either the ARM7TDI or the PXA271.

The MIRACL Crypto SDK [22] is an open-source Elliptic Curve Crypto SDK that supports many different platforms. It is a C library with some field arithmetic in assembly for many of its supported the platforms. Some timings for this library can be found in [21] and are also listed in Table IV.

The RELIC toolkit [23] is an open-source cryptographic library that supports many different architectures.

TABLE IV
TIMINGS FOR POINT MULTIPLICATIONS. ALL TIMINGS ARE GIVEN IN MILLISECONDS AND ENERGY IS GIVEN IN MICROJOULES (µJ). THE ATMEGA128L RUNS AT 7.37MHZ EXCEPT WHEN INDICATED WITH ^{a}, THE MSP430 RUNS AT 8.192MHZ, THE ARM7TDMI RUNS AT 80MHZ, AND THE ARM CORTEX-M0+ RUNS AT 48MHZ.

Platform	Author	Curve	Multiply [ms]	Multiply [µJ]
ARM7TDMI	MIRACL [21]	P-192	38^r	182.4^e
ARM7TDMI	MIRACL [21]	P-224	53^r	254.4^e
ATMega128L	Aranha et al. [13]	K-163	320^r	9600^e
ATMega128L^a	Kargl et al. [10]	167-bit^b	763^r	24840^e
ATMega128L	Aranha et al. [13]	K-233	730^r	21900^e
Cortex-M0+	*This work kG*	*sect233k1*	*39.70^f*	*20.63^m*
Cortex-M0+	*This work kP*	*sect233k1*	*59.18^r*	*34.16^m*
Cortex-M0+	Relic kP	K-233	115.7^r	69.48^m
Cortex-M0+	Relic kG	K-233	117.1^f	70.26^m
MSP430F1611	NanoECC [8]	P-160	720^f	8847^m
MSP430F1611	NanoECC [8]	K-163	1040^f	12780^m

^{a} Runs at 8MHz.
^{m} Energy values obtained by measurement.
^{e} Energy values obtained by estimation.
^{f} Fixed-point multiplication.
^{p} A custom prime curve is used.
^{r} Random point multiplication.

TABLE V
AVERAGE CYCLE TIMES FOR MODULAR MULTIPLICATION AND MODULAR SQUARING ON DIFFERENT PLATFORMS.

Author	Platform	Word size	Sqr	Mul	Field
S. Erdem [14]	ARM7TDMI	32	348	4359	$\mathbb{F}_{2^{228}}$
S. Erdem [14]	ARM7TDMI	32	389	5398	$\mathbb{F}_{2^{256}}$
Aranha et al. [13]	ATMega128L	8	570	4508	$\mathbb{F}_{2^{163}}$
Aranha et al. [13]	ATMega128L	8	956	8314	$\mathbb{F}_{2^{233}}$
Kargl et al.[10]	ATMega128L	8	-	2593	$\mathbb{F}_{160}$
Kargl et al.[10]	ATMega128L	8	663	5490	$\mathbb{F}_{2^{167}}$
P. Szczechowiak et al. [12]	ATMega128L	8	1581	13557	$\mathbb{F}_{2^{271}}$
Gouvea [15]	MSP430X ^{a}	16	630	741	$\mathbb{F}_{160}$
Gouvea [15]	MSP430X ^{a}	16	199	3585	$\mathbb{F}_{2^{163}}$
Gouvea [15]	MSP430X ^{a}	16	1369	1620	$\mathbb{F}_{256}$
Gouvea [15]	MSP430X ^{a}	16	325	8166	$\mathbb{F}_{2^{283}}$
This work	*Cortex-M0+*	32	*395*	*3672*	$\mathbb{F}_{2^{233}}$
TinyPBC [11]	PXA271	32	187	2025	$\mathbb{F}_{2^{271}}$
TinyPBC [11]	PXA271 ^{b}	32	187	1411	$\mathbb{F}_{2^{271}}$

^{a} This model has a long 32-bit multiplier
^{b} This model (wMMX) has a SIMD instructions set.

In the following text we will present two implementations. First, we will present an implementation that relies exclusively on the RELIC toolkit to make all its computations. Next, we present an implementation that was largely developed in C and assembly, but also makes use of the RELIC toolkit to perform some calculations. The curve and algorithmic parameters for both implementations were chosen to match each other as close as possible.

1) RELIC implementation: The RELIC toolkit was used to make an implementation with the following configuration: A binary Koblitz curve of order 2^{233} is used. The left-to-right wTNAF method with $w = 4$ was used for point multiplication. Point additions are done in mixed LD-affine coordinates. Fast reduction is done because the reduction polynomial is trinomial. Inversion is performed with the Extended Euclidean algorithm and squaring is done using the table-based method. The easy-to-understand arithmetic option is selected but this could be replaced with calls to the GMP library. However, we were unable to get the GMP library cross-compiled for the ARM Cortex-M0+ and were therefore unable to test this feature.

The RELIC implementation was measured to have an average power consumption of 600 µW while performing a random point multiplication, and 600.5 µW while performing a fixed point multiplication. This implementation requires an average of 5621045 cycles, and 72.5 µJ for a random point multiplication, and only 5553828 cycles, and 71.6 µJ for a fixed point multiplication. The average cycle time and energy usage of this implementation is compared to others in Table IV.

2) Proposed implementation: An implementation was made using C and assembly. The binary Koblitz sect233k1 curve was used. For point multiplication the left-to-right wTNAF method was used. A value of $w = 4$ was used for random point multiplication (kP), and $w = 6$ for fixed point multiplication (kG). Point additions are done in mixed LD-affine coordinates. The RELIC toolkit was used to perform the TNAF precomputation, and TNAF transformation of the scalar k. The `LD with fixed registers` method was used for field multiplication, fast reduction was done one word at a time, inversion was done with the Extended Euclidian algorithm, and squaring was done with the table-based method. All the field arithmetic was

implemented in C and assembly.

Our proposed implementation was measured to have an average power consumption of 577.2 µW for a random point multiplication, and 519.6 µW for a fixed-point multiplication. On average our implementation of the random point multiplication requires 2814827 cycles, and 36.6 µJ, whereas our fixed point multiplication requires 1864470 cycles, and 24.6 µJ. Therefore our implementation has a cycle count that is 1.99 times faster than the RELIC implementation for random point multiplication, and has a cycle count which is 2.98 times faster than the RELIC implementation for fixed point multiplication. The average execution time and energy usage of this implementation is compared to others in Table IV and Table V. The C and assembly implementations' cycle times for the field arithmetic routines are shown in Table VI. The accumulated execution time for different operations are shown in Table VII for both a random point multiplication (kP), as well as a fixed point multiplication (kG).

TABLE VI
AVERAGE CYCLE TIMES FOR FIELD ARITHMETIC ALGORITHMS IN $\mathbb{F}_{2^{233}}$.

Operation	C language	Assembly
Modular Squaring	419	395
Inversion	141 916	-
`LD with rotating registers`	5 592	-
`LD with fixed registers`	5 964	3 672
kP	3 516 295	2 761 640
kG	2 494 757	1 864 470

TABLE VII
TOTAL ACCUMULATED TIMINGS PER OPERATION FOR RANDOM POINT MULTIPLICATION (kP), AND FIXED POINT MULTIPLICATION (kG).

Operation	*kP*	*kG*
TNAF Representation	178 135	185 926
TNAF Precomputation	398 387	0
Multiply	1 108 890	821 178
Multiply Precomputation	249 750	184 950
Square	362 379	342 294
Inversion	139 936	139 656
Support functions	377 350	376 392
Total	2 814 827	1 864 470

V. FUTURE WORK

The current implementation doesn't execute in constant-time and is therefore at risk of a power analysis attack. It would be very interesting to see the results of an implementation where the point multiplication routine is implemented in constant-time by using an algorithm like the Montgomery-Ladder [24] method.

VI. CONCLUSION

We made an ECC implementation based on a binary Koblitz curve in $\mathbb{F}_{2^{233}}$, because we estimated that it will lead to a faster implementation with a lower energy consumption than a prime curve implementation with an equivalent security level. A new field multiplication algorithm was proposed, called the `López-Dahab with fixed registers`, which is an optimization of the `López-Dahab` (`LD`) algorithm. Our proposed algorithm has a cycle count of 3672 cycles, and a performance improvement of 15% over the `LD with rotating registers` algorithm. Our implementation of a random point multiplication requires 2814827 cycles, and 34.16 µJ, whereas our fixed point multiplication requires 1864470 cycles, and 20.63 µJ. The energy consumption of our implementation beats all known software implementations on embedded platforms, of a point multiplication, on the same equivalent security level by a factor of 7.4.

REFERENCES

[1] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[2] David W. Kravitz, "Digital signature standard," *U.S. Patent No. 5,231,668*, 1993.

[3] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Jan. 1987.

[4] V. S. Miller, "Use of elliptic curves in cryptography," in *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*. New York, NY, USA: Springer-Verlag New York, Inc., 1986, pp. 417–426. [Online]. Available: http://dl.acm.org/citation.cfm?id=18262.25413

[5] ARM, "Cortex-M0+ Technical Reference Manual, Revision: r0p1," 2012, accessed: 2013-06-06.

[6] I. Freescale Semiconductor, "MKW01Z128, Highly-integrated, cost-effective single-package solution for sub-1 GHz applications, Rev. 3, 5/7/2013," 2013, accessed: 2013-06-06.

[7] J. López and R. Dahab, "High-speed software multiplication in $\mathbb{F}_{2^m}$," in *Progress in Cryptology-INDOCRYPT 2000*. Springer, 2000, pp. 203–212.

[8] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks." in *EWSN*, ser. Lecture Notes in Computer Science, R. Verdone, Ed., vol. 4913. Springer, 2008, pp. 305–320. [Online]. Available: http://dblp.uni-trier.de/db/conf/ewsn/ewsn2008.html#SzczechowiakOSCD08

[9] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," in *Cryptographic Hardware and Embedded Systems-CHES 2004*. Springer, 2004, pp. 119–132.

[10] A. Kargl, S. Pyka, and H. Seuschek, "Fast arithmetic on ATmega128 for elliptic curve cryptography," *International Association for Cryptologic Research Eprint archive*, 2008.

[11] L. B. Oliveira, M. Scott, J. López, and R. Dahab, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," in *In Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on*, 2008, pp. 173–180.

[12] P. Szczechowiak, A. Kargl, M. Scott, and M. Collier, "On the application of pairing based cryptography to wireless sensor networks," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 1–12.

[13] D. F. Aranha, L. B. Oliveira, J. López, and R. Dahab, "Efficient implementation of elliptic curve cryptography in wireless sensors," *Advances in Mathematics of Communications*, vol. 4, no. 2, pp. 169–187, 2010.

[14] S. S. Erdem, "Fast software multiplication in $\mathbb{F}_2[x]$ for embedded processors," *Turkish Journal of Electrical Engineering & Computer Sciences*, 2012.

[15] C. P. L. Gouvêa, L. B. Oliveira, and J. López, "Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller," *J. Cryptographic Engineering*, vol. 2, no. 1, pp. 19–29, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/jce/jce2.html#GouveaOL12

[16] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM systems journal*, vol. 29, no. 4, pp. 526–538, 1990.

[17] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, vol. 7, 1963, p. 595.

[18] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[19] K. Piotrowski, P. Langendoerfer, and S. Peter, "How public key cryptography influences wireless sensor node lifetime," in *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, ser. SASN '06. New York, NY, USA: ACM, 2006, pp. 169–176. [Online]. Available: http://doi.acm.org/10.1145/1180345.1180366

[20] D. G. Chinnery and K. Keutzer, "Closing the power gap between ASIC and custom: an ASIC perspective," in *Proceedings of the 42nd annual Design Automation Conference*. ACM, 2005, pp. 275–280.

[21] Certivox Ltd., "Benchmarks and Subs," https://wiki.certivox.com/display/EXT/Benchmarks+and+Subs?sortBy=createddate, Oct. 2012, accessed: 2013-05-28.

[22] ——, "MIRACL Cryptographic Library," https://certivox.com/solutions/miracl-crypto-sdk/, 2013, accessed: 2013-05-28.

[23] D. Aranha and C. P. L. Gouvêa, "RELIC Cryptographic Toolkit," https://code.google.com/p/relic-toolkit/, May 2013, accessed: 2013-05-28.

[24] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of computation*, vol. 48, pp. 243–264, 1987.