8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors

Erich Wenger, Thomas Unterluggauer, and Mario Werner

Graz University of Technology
Institute for Applied Information Processing and Communications
Erich.Wenger@iaik.tugraz.at,
{T.Unterluggauer,M.Werner}@student.tugraz.at

Abstract. The decision regarding the best suitable microprocessor for a given task is one of the most challenging assignments a hardware designer has to face. In this paper, we make a comparison of cycle-accurate VHDL clones of the 8-bit Atmel ATmega, the 16-bit Texas Instruments MSP430, and the 32-bit ARM Cortex-M0+. We investigate their runtime, chip area, power, and energy characteristics regarding Elliptic Curve Cryptography (ECC), one of the practically most resource-critical public-key cryptography systems. If ECC is not implemented with greatest care, its implementation can lead to excruciating runtimes or enable practical side-channel attacks. Considering those important requirements, we present a constant runtime, side-channel protected, and resource saving scalar multiplication algorithm. To tap the full potential of all three microprocessors, we perform assembly optimizations and add carefully crafted instruction-set extensions. To the best of our knowledge, this is the first thorough software and hardware comparison of these three embedded microprocessors.

Keywords: ATmega, MSP430, Cortex-M0+, Elliptic Curve Cryptography, Instruction-Set Extension, Software and Hardware Evaluation.

1 Introduction

Motivation. It is a well-known fact that embedded microprocessors play a significant role within a huge number of consumer, industrial, commercial and military applications. Microprocessors are being produced and deployed in huge numbers and are the beating heart of, e.g., smart cards, wireless sensor networks, or in future even RFID tags. Those applications require solutions that are highly optimized in order to be cheap, energy-efficient, and/or power-efficient, while being versatile and delivering the necessary performance.

To meet all these requirements, the high demands of security and cryptography have too often been disregarded. Especially public-key cryptography needs to be implemented with great care in order to achieve small, performant, and energy-saving solutions. Since RSA and ElGamal based crypto systems simply require too much memory, Elliptic Curve Cryptography (ECC) seems to be the best choice. However, ECC is a highly demanding challenge within most applications, and therefore the decision regarding the most suitable microprocessor usually is the most discussed topic within a hardware manufacturer. To evaluate the performance of ECC in software and in hardware, we built VHDL clones of three of the most popular microprocessors.

Related Work. The research community recognized the challenge of efficiently implementing ECC. In this context, we want to cite [19, 20, 22, 33, 42, 45], just to name a few. Those papers presented and used a lot of different approaches to improve the performance of ECC on embedded microprocessors. Unfortunately, in many papers, the authors sacrifice practical crucial properties, e.g., the memory footprint or practical side-channel security threats for the sake of fastest runtimes. Other characteristics like power and energy consumptions are also often neglected. Szczechowiak et al [42] is a welcome exception as they presented measured power values for the ATmega and the MSP430 microprocessor. However, how comparable are those values as both processors were manufactured in different ASIC technologies by different vendors? A fair comparison of the investigated microprocessors must utilize a common design flow, common technologies, and practically secured software implementations.

Our Contribution. In this paper, we perform a systematic and comprehensive approach to evaluate ECC on cycle-accurate VHDL clones[1] of three of the most popular microprocessors: the 8-bit Atmel AVR ATmega, the 16-bit Texas Instruments MSP430, and the 32-bit ARM Cortex-M0+. Our contribution is composed of the following points:

- We derive a point multiplication methodology from previous work which is light-weight and secure against (most) side-channel attacks. The resulting algorithm can be applied to any future embedded designs.
- All our software implementations for the three processors are secure against side-channel attacks and highly optimized using state-of-the-art multi-precision integer multiplication techniques. Runtime, chip area, power, and energy results are given for four different standardized elliptic curves (`secp160-192-224-256r1`).
- We built three cycle-accurate clones of three of the most popular microprocessors and evaluate them in an 130 nm ASIC manufacturing process. The hardware models are based on publicly available design documents and software simulators. It is quite unlikely that Atmel, Texas Instruments, or ARM would have given us their cores for such a comparison. Their chips are produced in different technologies, so any comparison of actual chips is impracticable for our purposes. Regarding code quality, we want to emphasize that Atmel, Texas Instruments, and ARM use similar libraries and tools as we do. Therefore our designs are probably very close to the real deal.
- We are the first to integrate instruction-set extensions (ISE) in actual clones of those microprocessors. The only common denominator of those three processors is the 16-bit instruction-set. In every other key aspect, they differ (e.g.

[1] Closed source for now, done by the authors.

8, 16, 32 bit datapaths, Harvard/Von Neumann architecture, ...). Therefore the multiply-accumulate ISEs have to be carefully crafted for each CPU core.
- We are the first to optimize ECC on the Cortex-M0+.
- Our results represent state-of-the-art of side-channel protected, fast, lightweight, and standardized asymmetric cryptography for embedded processors.

The paper is structured as follows: Section 2 presents and analyzes the side-channel protected elliptic-curve point multiplication algorithm. Within Section 3 the three processors are reviewed and compared on an architectural level. Sections 4 and 5 summarize the assembly and instruction-set optimizations. A rigorous analysis of all implementation results is performed in Section 6. Section 7 concludes the paper.

2 Elliptic Curve Cryptography

When implementing elliptic curve cryptography, a designer has a multitude of options. In the following we present an algorithm for the point multiplication which was chosen based on four characteristics:

- It is easy to **tamper** with embedded microprocessors. Timing attacks, power-analysis attacks and fault-analysis attacks are a real and omnipresent danger. For that matter we will not claim to be secure against all kinds of attacks, but by choosing a methodology that is aware of many kind of attacks, we emphasize the practical significance of the results presented later.
- ECC is a **feature**. Unlike, e.g., the work in [42] (Comb method with window size $w = 4$), we think that only a small fraction of the available program and data memory resource should be used for ECC so that the actual application is not hindered in its operation. Therefore we choose a point-multiplication formula which does not allocate the whole memory for pre-computed or temporary points. Reduced memory requirements further allows hardware designers to save money by equipping the chip with smaller memories.
- Standards were made to be used and simplify the **interoperability** of products. Thus, by choosing a NIST [35] or SECG [7] standard, any company can be sure that their product is compatible with products from other vendors.
- Achieving a **high speed** is an ubiquitous goal of nearly every designer. By getting the most out of an available hardware, one reduces latency times (important in real-time protocols) and saves energy (important for battery-powered devices and from an economic point of view). As we do not sacrifice our security requirements for speed, we concentrate on improving the finite-field operations by doing assembly and instruction-set optimizations.

In Algorithm 1, we present the point multiplication formula used for all our practical evaluations. A detailed analysis of Algorithm 1 is given in Appendix A. Our goal was to design an algorithm which can be used for Diffie-Hellman key exchanges (DHKE) and elliptic-curve based signatures (ECDSA [35]), which are the major features embedded applications actually require. The algorithm

Algorithm 1 Elliptic curve point-multiplication algorithm used for evaluation.

```
Input: Domain parameters, secret scalar $k$ with MSB $= 1$, point $P = (x, y)$.
Output: $R = k \times P$
 1: if $y^2 \neq x^3 + ax + b$ then Perform Error Handling
 2: $(X, Y, Z) \leftarrow (x \cdot \lambda, y \cdot \lambda, \lambda)$    ▷ Randomize Projective Coordinates
 3: if $Y^2 Z \neq X^3 + aXZ^2 + bZ^3$ then Perform Error Handling
 4: $Q[0] \leftarrow (X, Z), Q[1] \leftarrow 2 \cdot (X, Y, Z)$    ▷ Initial Point Doubling
 5: for $i = |k| - 2$ downto $0$ do    ▷ Montgomery Ladder
 6:     $Q[k_i] \leftarrow Q[k_i] + Q[k_i \oplus 1]$
 7:     $Q[k_i \oplus 1] \leftarrow 2 \cdot Q[k_i \oplus 1]$
 8: end for
 9: $(X, Y, Z) \leftarrow$ y-recovery$(Q[0], Q[1])$
10: if $Y^2 Z \neq X^3 + aXZ^2 + bZ^3$ then Perform Error Handling
11: $R = (x, y) \leftarrow (XZ^{-1}, YZ^{-1})$
12: if $y^2 \neq x^3 + ax + b$ then Perform Error Handling
```

is using a Montgomery ladder [34] with Randomized Projective Coordinates (RPC) [10] and multiple point validation (PV) checks. After an initial PV check the coordinates are randomized. In step 3, the point is again checked within the projective coordinates. A fault attack on the randomized projective coordinates is much more complex. Then, an initial point doubling within the RPC is performed. The double of the original point is needed for the following Montgomery ladder. Here we use the common-z interleaved point addition and doubling formulas by Hutter, Joye, and Sierra [25]. As this is the most costly part of the algorithm, no PV checks are performed within it. For the following recovery of the y-coordinates, both $Q[0]$ and $Q[1]$ are used. Another two PV checks are performed before and after the inversion of the Z-coordinate. One may argue that several of the PV checks are redundant, but because they hardly have any impact on the speed, we perform them anyways.

Runtimes of all finite-field operations are constant and data-independent. Therefore, a finite-field inversion was implemented based on Fermat's little theorem (inversion by exponentiation). Particularly, the algorithm is based on the exponentiation trick by Itoh and Tsujii [27]. Although this trick is usually applied to elliptic curves over binary fields, we utilize it to optimize the inversion for the standardized Mersenne-like primes, nearly halving the number of multiplications needed for an exponentiation with a fixed exponent.

Summarizing, it is important to utilize the available resources. Therefore a detailed knowledge of the used microprocessors is necessary to achieve competitive results. Section 3 discusses the characteristics of the investigated embedded microprocessors.

3 Microprocessor Architectures

This paper focuses on three of the most popular embedded microprocessors: the 8-bit Atmel ATmega AVR, the 16-bit Texas Instruments MSP430 and the 32-

bit ARM Cortex-M0+ microprocessors. All of them were designed for embedded applications, in which price and power consumption matter more than the maximum clock speed or the amount of available data or program cache. In fact, those RISC processors do not have any cache. In this section, we introduce the three processor architectures and discuss their capabilities relevant for ECC.

Atmel ATmega AVR series. In 1996, two students from the Norwegian Institute of Technology developed the first AVR processor. Today, designers can choose from a vast range of descendants. Especially the ATmega series [2] has been and is used in a magnitude of commercial products.

The ATmega is a 8-bit RISC processor with separated program, data, and I/O memory buses (Harvard architecture). It comes with 32 general-purpose registers (GPR) and 91 (133 including simulated) instructions. To perform integer arithmetic, operands need to be loaded (2 cycles) to the GPRs, processed within the GPRs, and stored back (2 cycles) to the data memory. A for multi-precision integer arithmetic [9] interesting 8-bit multiply-accumulate operation (`LD`, `LD`, `MUL`, `ADD`, `ADC`, `ADC`) takes 9 cycles.

Texas Instruments MSP430 series 1. One of the most successful, direct competitor of the ATmega is the MSP430 processor series [43] by Texas Instruments. With its six low-power modes it is most interesting for low power, and low-energy applications. This is why it is used for many wireless sensor nodes such as the EPIC Mote, TelosB, T-Mote Sky, and XM1000 platforms.

The original series-1 MSP430 is a 16-bit RISC processor with a single combined data and program bus. Merely 12 of its 16 16-bit registers are actually usable as general-purpose registers. The MSP430 series comes with a fully orthogonal instruction set of only 27 instructions with 7 addressing modes. Unfortunately, there is no dedicated multiplication instruction, but a memory mapped $16 \times 16 \rightarrow 32$-bit multiplier, with multiply-accumulate feature, is available. Despite the high costs introduced by transfering the operands from data memory to the multiplier, a 16-bit multiply-accumulate operation (`MOV`, `MOV`, `NOP`, `ADD`) can be performed in 13 cycles.

ARM Cortex-M0+ series. In the recent years, ARM made a name for itself with supplying smart phones and tablets with powerful energy-saving processors, namely the Cortex-A series. For embedded applications however, Cortex-M series processors are more suitable. The Cortex-M0+ embedded microprocessor [1] is the smallest, most energy-efficient ARM ever built and supports a subset of the Thumb-2 instruction-set. This 32-bit RISC processor is designed as direct competitor for the ATmega and MSP430 processors. Launched in 2012, major companies (e.g., Freescale [15], Fujitsu [16], or NXP [36]) just started to introduce the Cortex-M0+ to their lineups.

Similar to the MSP430, the Cortex-M0+ comes with a Von Neumann architecture. Its 32-bit address and data buses enable the addressing of up to 4 GByte of data, preparing it perfectly for future memory requirements. Exactly 13 of its 16 32-bit registers can be used as general-purpose registers, but most of its 56 instructions can only access the lower 8 registers `R0`–`R7`. Registers `R8`–`R15` are accessible through a `MOV` instruction only. Optionally, the Cortex-M0+ comes with

Table 1. Summary of the embedded microprocessors.

Characteristic	**ATmega**	**MSP430** (1 Series)	**Cortex-M0+**
Data-Width	8 bits	16 bits	32 bits
Instruction-Word Size	16 bits	16 bits	16 bits
Architecture	Harvard	Von Neumann	Von Neumann
General-Purpose Registers	32	12/16	8/13/16
Number of Instructions	81	27	56
Max. Data Memory	16 kByte	10 KByte	4 GByte
Max. Program Memory	256 kByte	60 KByte	4 GByte
Multiply Accumulate[a]	9 cycles	13 cycles	29 cycles
Used Clone	JAAVR [47]	IDLE430 [50]	Xetroc-M0+ [44]
Core area[b]	6,140 GE	4,913 GE	15,262 GE
Registers	2,002 GE	1,732 GE	4,176 GE
Multiplier	372 GE	1,751 GE[c]	2,766 GE

[a] Load two operands, multiply and accumulate them.

[b] Including memory arbiter and necessary special function registers.

[c] Memory mapped and therefore not part of the core area.

a bit-serial or a single-cycle 32 × 32 → 32-bit multiplication instruction. Note that for ECC, also the upper half of the product is necessary for multi-precision integer arithmetic. In the following, we use this multiplier as 16 × 16 → 32-bit multiplier. Section 4 illustrates the implementation of an efficient 29-cycle 32-bit multiply-accumulate operation.

Summary of the Embedded Microprocessors. The common denominators of the three embedded microprocessors are that they are RISC processors, support single-cycle register-to-register operations, and have 16-bit instruction sets (with some 32-bit exceptions). The major differences are summarized in Table 1. The three microprocessors utilize different architectures, have different amounts of available registers, clearly distinct instruction sets and support different amounts of data and program memory. Those differences become apparent when the actual hardware footprint is evaluated. Most remarkably, the 16-bit MSP430 (4,913 GE) requires less chip area than the 8-bit ATmega (6,140 GE). This is the price the ATmega has to pay for its three memory buses and the vast instruction set. To efficiently perform integer multiplications, the MSP430 additionally needs a memory mapped multiplier which is 1,751 GE in size. Compared to the ATmega and the MSP430, the 32-bit Cortex-M0+ is much larger. It requires 15,262 GE in a configuration with a single-cycle 32-bit multiplier. The optional 32-cycle bit-serial multiplier saves 1,363 GE which brings the Cortex-M0+ to a minimum size of 13,899 GE in the used 130 nm UMC process.

Related Work. ARM specifies that their Cortex-M0+ is only 12 kGE large in a 90 nm process. When synthesizing our clone in a 90 nm UMC process it requires 12,436 GE. So in terms of area, our Cortex-M0+ is (as aimed for) very close to the original. As neither Atmel nor Texas Instruments released characteristics of their processors, we can only compare our clones to the versions uploaded to `opencores.org`. Compared to the openMSP430 [37], our MSP430 is $5,958 - 4,913 = 1,045\,\mathrm{GE}$ smaller. Compared to other (insufficiently tested) ATmega or AVR clones, our ATmega clone is similarly small.

4 Assembly Optimizations for ECC

As we already fixed the point arithmetic, we focus our effort on the finite-field operations. Most crucial is the finite-field multiplication as it contributes to 90 % of the runtime of a point multiplication. Hence, optimizing the runtime of the finite-field multiplication automatically improves the runtime of the point-multiplication algorithm. Additionally, it leads to a speedup of the finite-field squaring, exponentiation and inversion operation. To optimize the finite-field multiplication, one can either perform assembly optimizations or extend the instruction-set (see Section 5).

While the currently fastest multi-precision multiplication approaches for the ATmega and the MSP430 are based on related work, it was necessary to come up with a new technique for the Cortex-M0+ as it has not yet been investigated.

ATmega. In 2004, Gura *et al.* [22] presented an efficient multi-precision multiplication method and applied it to the ATmega. Hutter and Wenger [26] presented the “Operand Caching” method in 2011. It further reduced the number of memory load operations at the cost of some memory store operations. As it fully utilizes the available general purpose registers as caches and currently is one of the fastest ways to perform multi-precision multiplications on an ATmega, we used their technique.

MSP430. The MSP430 only has 12 useable GPRs, of which three registers have to be used as pointers and further three registers are necessary for the accumulation of intermediate results. With the remaining six registers, we applied the product-scanning technique by Comba [9]. This technique fully utilizes the multiply-accumulate functionality of the memory-mapped multiplier. It was first described by Gouvêa and López [19] and is fully tailored to the MSP430.

Cortex-M0+. ECC performance of the Cortex-M0+ has never been examined before. Most notable are the works of Aydos *et al.* [3], who optimized ECC for an ARM7TDMI processor, and Bernstein and Schwabe [4], who optimized the NaCl cryptographic library for a Cortex-A8. However, none of their work had to deal with the limitations of a Cortex-M0+: Its multiplier only computes 32-bit products, most instructions are restricted to registers `R0`–`R7`, and, for most instructions, the destination register must equal one of the source registers, i.e. in each multiplication one of the operands is overwritten by the product.

We evaluated several multiplication techniques and finally settled for a product-scanning multi-precision multiplication method. Its centerpiece is shown in Algorithm 2: the two operand references are moved from registers `R8` and `R9` to `R1` and `R2`. Consequently, we can load their values from the memory. Then, five registers are available to perform four $16 \times 16 \rightarrow 32$-bit multiplications (steps 9–13), whereby the 16-bit masking steps are performed only once (steps 5–7). Steps 14–27 accumulate the four 32-bit products into the registers `R3`–`R5`.

Algorithm 2 Multiply-Accumulate Operation on Cortex-M0+.

```
Input: R8 and R9 are pointers.
Output: R3, R4, R5 contain the sum.
MOV  R1, R8
LDR  R1, [R1, #offset1]
MOV  R2, R9
LDR  R2, [R2, #offset2]

UXTH R6, R1
UXTH R7, R2
LSR  R1, R1, #16
LSR  R2, R2, #16

MOV  R0, R6
MUL  R0, R0, R7        ▷ Low–Low
MUL  R6, R6, R2        ▷ Low–High
MUL  R2, R2, R1        ▷ High–High
MUL  R1, R1, R7        ▷ High–Low
MOV  R7, #0
ADD  R3, R3, R0        ▷ Low–Low
ADC  R4, R4, R2        ▷ High–High
ADC  R5, R5, R7

LSL  R0, R6, #16       ▷ Low–High
LSR  R2, R6, #16
ADD  R3, R3, R0
ADC  R4, R4, R2
ADC  R5, R5, R7

LSL  R0, R1, #16       ▷ High–Low
LSR  R2, R1, #16
ADD  R3, R3, R0
ADC  R4, R4, R2
ADC  R5, R5, R7
```

The stack is used to temporarily store the product of the multi-precision integer multiplication. Hence, we benefit from addressing relative to the stack pointer and avoid moving the address to one of the registers `R0`–`R7`. In a second step, a reduction is performed by taking advantage of the Mersenne-like primes.

Assembler Optimized Results. We did all assembly optimizations for four standardized elliptic curves (`secp160-192-224-256r1`). In order to keep the general view, Table 2 depicts the memory footprints, the runtimes for finite-field operations and the point multiplication over the `secp160r1` elliptic curve.

In terms of **ROM** size the processors behave converse their word sizes. The implementation for the ATmega takes up 7,762 bytes in ROM, which is twice as much as for the Cortex-M0+. This is mainly a result of the unrolled integer multiplication. The MSP430 behaves well as it requires only 13% more ROM than the Cortex-M0+. With respect to **RAM**, the ATmega (402 byte) and the Cortex-M0+ (404 byte) perform similarly, consuming about 40% more RAM than the MSP430 (290 byte). The increased RAM footprint of the ATmega is due to the elliptic curve constants that need to be loaded to the RAM at startup. The root of the increased memory usage of the Cortex-M0+ lies within its calling hierarchy. Each `PUSH` operation stores four bytes of data within the RAM. These facts combined make the MSP430 an economic platform in terms of memory footprint and chip area.

Impressingly, the **finite-field addition** on the Cortex-M0+ is 2.6 times faster than an addition on the MSP430. The main reason for that is the load-multiple `LDM` instruction of the Cortex-M0+, which allows loading multiple words into several registers requiring only $\#words + 1$ cycles. The load `LDR` instruction takes 2 cycles per word and therefore $2 \times \#words$ cycles would be needed.

Table 2. Benchmark data of assembly optimized implementations for `secp160r1`.

Processor	**ROM**	**RAM**	**Addition**	**Mult.**	**Inversion**	**Point Mult.**	**Core Area**
	[Bytes]	[Bytes]	[Cycles]	[Cycles]	[Cycles]	[Cycles]	[kGE]
Atmega	8,358	402	291	3,024	519,217	9,230,048	6,140
MSP430	4,788	290	163	1,905	327,366	5,779,957	7,003
CortexM0+	4,256	404	62	942	162,500	2,809,619	15,262
Relative Performance							
Atmega	1.96	1.46	4.69	3.21	3.20	3.29	1.00
MSP430	1.13	1.00	2.63	2.02	2.01	2.06	1.14
CortexM0+	1.00	1.39	1.00	1.00	1.00	1.00	2.49

As the runtime of **integer multiplication** scales quadratically, one expects the 32-bit Cortex-M0+ to be four times faster than the 16-bit MSP430 and the 16-bit MSP430 to be four times faster than the 8-bit ATmega. But they are not. The Cortex-M0+ is only twice as fast as the MSP430. The reason for that is the tremendous overhead needed to perform a $32 \times 32 \rightarrow 64$-bit multiplication using the $32 \times 32 \rightarrow 32$-bit integer multiplier (see the highly optimized Algorithm 2). But also the MSP430 is only 1.6 times faster than the ATmega. This is because the MSP430 has a memory mapped multiplier and the ATmega has an integrated multiplier.

By combining finite-field additions, multiplications, and inversions, the runtimes for **`secp160r1` point multiplications** were obtained. They are 9.2 million cycles for the ATmega, 5.8 million cycles for the MSP430, and 2.8 million cycles for the Cortex-M0+. As the finite-field multiplication contributes to the majority of this runtime, the ratios for the point multiplications are nearly identical to the ratios of the finite-field multiplication. Equipping the Cortex-M0+ with a bit-serial multiplier quadruples its runtime: With 11.9 million cycles the bit-serial multiplier simply defeates the purpose of having a 32-bit processor. Hence, we do not recommend implementing prime-field based ECC on a Cortex-M0+ without single-cycle multiplier. Consequently, instruction-set extensions were equipped to improve runtimes and to monitor how the performance ratios change.

5 Instruction-Set Modifications

We carefully crafted the VHDL clones of the ATmega, the MSP430, and the Cortex-M0+ to be cycle-compatible with its originals. During that process, we also observed some minor shortcomings regarding their respective potentials. This section is all about maximizing the performance by improving the runtime of existing instructions and adding multiply-and-accumulate [21] instructions. This `MULACC` instruction is optimal for multi-precision multiplication. It is used to multiply two n-bit registers and add the 2n-bit product to three accumulation registers. As the three processors differ significantly, `MULACC` had to be integrated differently for each processor.

ATmega. Our modifications of the ATmega are based on Wenger [46]. In this paper, we showed among other things how to improve load, store, and multiply

Table 3. Benchmarks of processors with instruction-set modifications for `secp160r1`.

Processor	**ROM**	**RAM**	**Addition**	**Mult.**	**Inversion**	**Point Mult.**	**Core Area**
	[Bytes]	[Bytes]	[Cycles]	[Cycles]	[Cycles]	[Cycles]	[kGE]
Atmega	5,828	402	176	984	170,053	3,268,486	7,039
MSP430	3,898	286	150	718	123,939	2,445,508	7,197
CortexM0+	3,088	408	62	369	64,859	1,231,946	18,700
Relative Performance of Modified Processors							
Atmega	1.89	1.41	2.84	2.67	2.62	2.65	1.00
MSP430	1.26	1.00	2.42	1.95	1.91	1.99	1.02
CortexM0+	1.00	1.43	1.00	1.00	1.00	1.00	2.66
Modified versus Assembly-optimized Implementations (Table 2)							
Atmega	−30.3%	±0.0%	−39.5%	−67.5%	−67.2%	−64.6%	14.6%
MSP430	−18.6%	−1.4%	−8.0%	−62.3%	−62.1%	−57.7%	2.8%
CortexM0+	−27.4%	1.0%	±0.0%	−60.8%	−60.1%	−56.2%	22.5%

instructions and execute them within a single cycle instead of two. Additionally, we added a single-cycle multiply-and-accumulate instruction, which was combined with the Operand-Caching multiplication method. In a special operating mode, activated by writing a memory-mapped configuration register, the existing `MUL` instruction is reinterpreted as `MULACC` instruction. Therefore, the software toolchain does not need to be modified. In fact, the trick of having a special mode for the instruction-set extension has also been applied for the MSP430 and Cortex-M0+.

MSP430. The advantage of the MSP430 is, that operands do not have to be explicitly loaded to core registers before their usage. The drawback is that the multiplier is only accessible via the memory. To get rid of this bottleneck, we removed the memory-mapped multiplier (saved 1,751 GE) and added a dedicated `MULACC` instruction within its core. Unfortunately, the 7 existing addressing modes were insufficient for our purposes. By perfectly utilizing the pre-existing auto-increment and a new auto-decrement addressing mode it is possible to load two operands, multiply-and-accumulate the data, and update the addressing registers. To omit manual pointer updates completely, we combined the new `MULACC` instruction with Wenger and Werner's [49] zig-zag product-scanning multiplication method. Other modifications with less impact on the ECC runtime improved move, jump, push, and call instructions by one to two cycles. This modifications do not only minimize the overhead of the C-calling convention, but also potentially improve the runtimes of any other algorithm run on the modified MSP430.

Cortex-M0+. As it is only possible to compute a $16 \times 16 \rightarrow 32$-bit product with the internal multiplier of the Cortex-M0+ and 29 cycles are necessary to perform a 32-bit (c.f. Algorithm 2) multiply-and-accumulate operation, it is specially important to equip the Cortex-M0+ with a `MULACC` extension. This extension reduced the for the product-scanning important sequence of `LDR`, `LDR`, `MULACC` instructions to mere 5 cycles. To save area, the pre-existing $32 \times 32 \rightarrow 32$-bit multiplier is reused and only extended to compute a 64-bit product. Therefore, the `MULACC` extension did only cost 3.4 kGE.

Results using Instruction-Set Modifications. In general, the core idea of all modifications was to improve the performance without adding unnecessary hardware. So the modifications of the ATmega (+14.8 %) and the Cortex-M0+ (+22.5 %) only marginally increased the size of the CPU cores. The effective size of the MSP430 only increased by 2.8 %. The slow, memory-mapped multiplier was approximately as large as the new dedicated datapath to multiply-and-accumulate within a single cycle. While the size of the CPU cores increased, the size of the program memory decreased by 19-30 %. The rather large unrolled integer multi-precision multiplication functions shrunk significantly and therefore the total chip areas actually decreased. However, the modifications only have very little impact on data memory utilization.

As intended, the modifications achieve a massive speedup of multiplications in the prime field (cf. Table 3). Throughout, the corresponding runtimes dropped by 60%, with the highest speedup achieved on the ATmega (-67%). As inversions are based on exponentiation to counteract side-channel attacks, the same impressing speedup is found there. Accordingly, point multiplication runtimes slumped by 65% on the ATmega and plunged by 57% on the others. Concerning addition, there are no performance gains for the Cortex-M0+ and the MSP430. Contrary to that, addition is performed 40% faster on the ATmega due to the improved timings of the load and store operations. Relating runtimes of the three modified processors, the Cortex-M0+ again achieves the best performance, being between 2-3 times faster than its competitors. However, its advantage diminishes slightly compared to the unmodified ATmega.

6 Discussion of Hardware Implementations

All our implementations for the three microprocessors, with and without instruction-set modifications, and over four elliptic curves were tested for correctness using externally generated test vectors, synthesized in a Faraday UMC 130 nm low-leakage ASIC library, placed-and-routed, and power-simulated (using Cadence RTL Compiler, Cadence Encounter). The huge number of results are accumulated within Table 4 and discussed in the following.

Memories. We used area-efficient single-port RAM macros as data memories and single-port Via-1 ROM macros as program memories. As their necessary sizes depend on the ECC implementation, they were chosen appropriately for each implementation. Experiments showed that synthesizing the program memories as standard logic cells resulted in smaller program memories after synthesis, but there were two problems: Firstly, it was virtually impossible to place and route the program memories without significantly decreasing the cell density, which actually increased the effective size of the program memory. Secondly, the ROM macros have a significantly lower power consumption compared to the synthesized program memories.

Runtime. The runtime is measured at 10 MHz and visualized in Figure 1. The time for a single, side-channel secured point multiplication varies between 123–923 ms. As expected, the 32-bit processor is faster than the 16-bit processor,

Table 4. Summary of all experiments.

Processor	**Version**	**Program**	**Data**[a]	**Chip Area**				**Power**	**Energy**	**Runtime**
		Memory	**Memory**	**ROM**	**RAM**	**Core**	**Total**	@10 MHz		@10 MHz
		[Bytes]	[Bytes]	[GE]	[GE]	[GE]	[GE]	[µW]	[µJ]	[ms]
`secp160r1`										
ATmega	ASM	8,358	402	11,807	3,754	6,140	21,701	545	503	923
MSP430	ASM	4,788	290	7,796	3,250	7,003	18,048	583	337	578
CortexM0+	ASM	4,256	404	8,270	4,308	15,262	27,840	718	202	281
ATmega	ISE	5,828	402	8,202	3,754	7,039	18,995	666	218	327
MSP430	ISE	3,898	286	6,363	3,225	7,197	16,786	794	194	245
CortexM0+	ISE	3,088	408	6,416	4,334	18,700	29,450	1,306	161	123
`secp192r1`, **NIST P-192**										
ATmega	ASM	10,238	462	11,807	4,107	6,140	22,054	556	839	1,509
MSP430	ASM	5,408	330	8,202	3,475	7,003	18,679	581	533	918
CortexM0+	ASM	4,860	448	8,270	4,560	15,262	28,092	716	329	459
ATmega	ISE	6,564	462	10,040	4,107	7,039	21,186	670	336	502
MSP430	ISE	4,142	330	7,796	3,475	7,197	18,468	801	283	353
CortexM0+	ISE	3,164	444	6,416	4,535	18,700	29,652	1,318	241	183
`secp224r1`, **NIST P-224**										
ATmega	ASM	12,570	526	15,484	4,485	6,140	26,109	571	1,326	2,321
MSP430	ASM	6,294	374	10,040	3,750	7,003	20,792	584	819	1,403
CortexM0+	ASM	5,672	496	8,270	4,838	15,262	28,369	716	496	693
ATmega	ISE	7,600	526	10,040	4,485	7,039	21,564	664	500	754
MSP430	ISE	4,588	370	7,796	3,725	7,197	18,718	805	419	521
CortexM0+	ISE	3,352	492	6,416	4,812	18,700	29,929	1,330	334	251
`secp256r1`, **NIST P-256**										
ATmega	ASM	16,112	590	17,029	4,838	6,140	28,006	548	1,914	3,493
MSP430	ASM	8,378	418	11,878	4,000	7,003	22,881	580	1,286	2,217
CortexM0+	ASM	7,168	540	10,123	5,089	15,262	30,475	719	771	1,073
ATmega	ISE	9,596	590	11,807	4,838	7,039	23,684	655	779	1,190
MSP430	ISE	6,168	416	10,040	3,975	7,197	21,212	791	717	907
CortexM0+	ISE	4,124	536	8,270	5,064	18,700	32,034	1,339	546	408

[a] Including Stack.

which in turn is faster than the 8-bit processor. Quite remarkably though is that the modified ATmega is nearly as fast as the native Cortex-M0+.

Area. The area visualized in Figure 1 accumulates the respective areas of the CPU, the ROM, the RAM, and core building blocks, such as an arbiter. Quite remarkably, the native and the modified MSP430 represent the smallest implementation, requiring around 16.8–18.0 kGE. Compared to that, the modified Cortex-M0+ (29 kGE) is 75 % larger.

Area-runtime-product. In the prestigious category of area-runtime-product, the modified implementations clearly outperform its native counterparts (see the dashed lines within Figure 1). The modified Cortex-M0+ system performs best, and the native ATmega system performs worst. However, the modified ATmega system provides a better performance for the used chip area than the native Cortex-M0+. Consequently, if some commercial company evaluates whether to switch to a more powerful Cortex-M0+, we can clearly recommend to replace the native MSP430 or ATmega with its modified counterpart,

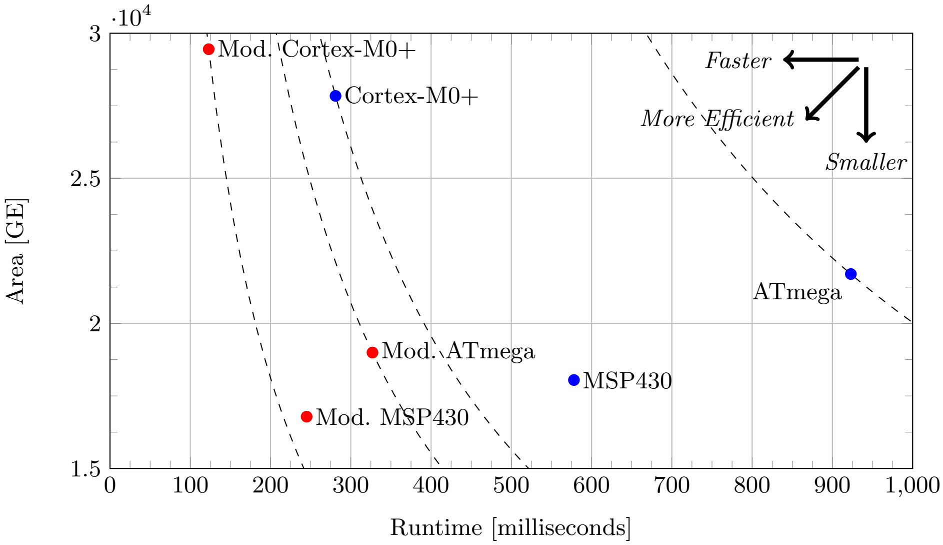

Fig. 1. Area-runtime-characteristics for `secp160r1` at 10 MHz.

presented within this paper. As nice side-effect, the software code-base does not have to be updated for a different processor.

Power. According to Figure 2, all designs require between 545–1,305 µW. The 8-bit ATmega requires the least amount of power, slightly less (6.5 %) than the MSP430. However, when their modified counterparts are compared, the modified ATmega needs 16 % less power than the modified MSP430. The Cortex-M0+ and the modified Cortex-M0+ need the most power.

Power-runtime-product: Energy. However, the same two processors shine within the energy-efficiency race. As represented by the dashed lines in Figure 2, the Cortex-M0+ based designs only need 161–202 µJ, while the other designs need 194–503 µJ. That is up to 60 % less. The MSP430 is 11–33 % more energy efficient than the ATmega.

Relating the Different Elliptic Curves. As initially stated and depicted in Table 4, we did not do our evaluation only with `secp160r1`, but also with `secp192r1`, `secp224r1`, and `secp256r1`. Most importantly, the results observed at the 80-bit security level are reproducible for the larger elliptic curves. On average, changing from one elliptic curve to the next larger one, costs 6 % of additional chip area, 53 % of additional runtime and 54 % of additional energy. The power consumption is not effected by the chosen elliptic curve.

Related Work. In terms of software implementations (c.f. Table 5), our implementations distinct themselves from related work with their low memory footprints and the side-channel countermeasures. In fact none of the implementations done by [20, 22, 33, 42, 45] have side-channel countermeasures built in. Therefore it is expected that, e.g., [20, 22, 42], achieve faster runtimes than we

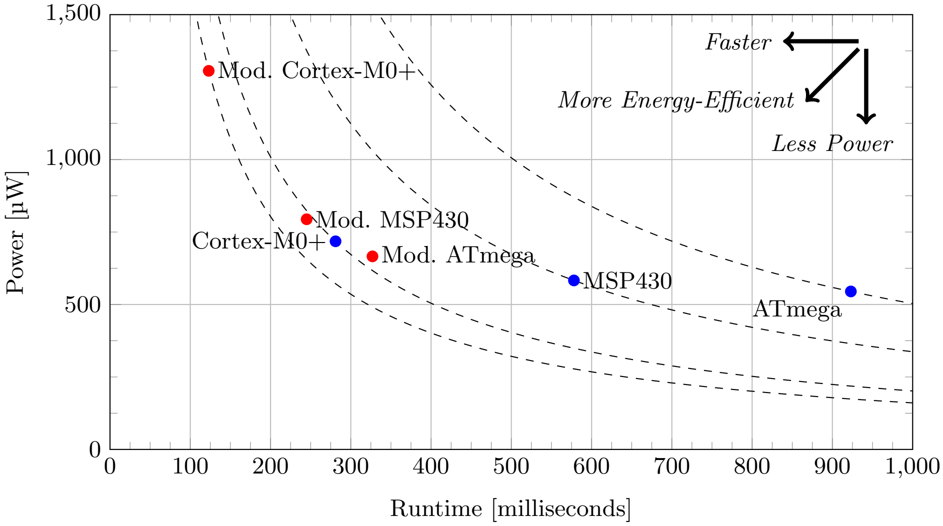

Fig. 2. Power-runtime-characteristics for `secp160r1` at 10 MHz.

do. However, e.g., [20, 42] need up to 7–10 times more program and data memory than we do.

For the sake of completeness, we also compare our modified processors with related hardware implementations (c.f. Table 6). Unfortunately, those dedicated hardware designs are faster than our flexible microprocessor based designs. However, in terms of chip area, our smallest modified MSP430 implementation is smaller than the work of [17, 24, 29, 38, 39]. Only the custom microprocessor design by Wenger *et al.* [48] is smaller. However, their microprocessor does not come with the vast (open-source) compiler toolchains, the ATmega, the MSP430, or the Cortex-M0+ provide.

7 Conclusion

In this work, three of the most popular micro-processors were evaluated regarding their runtime, chip area, power, and energy consumption on standard-compatible side-channel protected elliptic curve cryptography. By comparing them using a single design flow, the same application, and with a common technology, we achieve a fair comparison between the different architectures. Our work might help any system architect on their decision regarding best suitable processor, best suitable security level, and whether or not to implement hardware extensions. Our results show that the Cortex-M0+ is the fastest and most energy-efficient processor (e.g., ideal for Wireless Sensor Nodes), the MSP430 enables the smallest and least power consuming hardware design (e.g., ideal for RFID tags), and the ATmega gains the most performance when instruction-set modifications are applied (e.g., ideal for long-lived products that must be

Table 5. Comparison with related software implementations (80 bit security level).

Curve	ROM	RAM	Runtime
	[Bytes]	[Bytes]	[kCycles]
ATmega			
Custom [42]	46,100	1,800	9,376
`secp160r1` [33]	20,768	1,774	15,060
`secp160r1` [22]	3,682	282	6,480
Our `secp160r1`	8,358	402	9,230
MSP430			
Custom [42]	31,300	2,900	5,898
`secp160r1` [20]	23,300	2,800	2,528
`secp160r1` [33]	16,218	1,866	11,821
`secp160r1` [45]	12,500	1,300	28,080
Our `secp160r1`	4,788	290	5,780

Table 6. Comparison with related hardware implementations.

Implementation	Area	Runtime
	[GE]	[kCycles]
96-bit security level		
Satoh *et al.* [39]	29,655	4,165
Hutter *et al.* [24]	19,115	859
Wenger *et al.* [48]	11,686	1,377
80-bit security level		
Öztürk *et al.* [38]	30,333	545
Fürbass *et al.* [17]	23,656	500
Kern *et al.* [29]	18,247	512
Our Mod. ATmega	18,995	3,268
Our Mod. MSP430	16,786	2,446
Our Mod. Cortex-M0+	29,450	1,232

equipped with ECC). Any designer now has to define their own metric and weigh the characteristics with each other. To the best of our knowledge, such an comprehensive evaluation has not been done before.

Acknowledgments

This work has been supported in part by the Austrian Government through the research program FIT-IT under the project number 835917 (project NewP@ss) and by the European Commission through the ICT Programme under contract ICT-SEC-2009-5-258754 TAMPRES.

References

1. ARM. Cortex-M0+ Processor, 2013. Available online at `http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php`.
2. Atmel Corporation. megaAVR Microcontroller, 2013. Available online at `http://www.atmel.com/products/microcontrollers/avr/megaavr.aspx`.
3. M. Aydos, T. Yanik, and Ç . K. Koç. A High-Speed ECC-based Wireless Authentication Protocol on an ARM Microprocessor. In *ACSAC*. IEEE, 2000.
4. D. Bernstein and P. Schwabe. Neon crypto. *CHES*, 2012.
5. I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *CRYPTO*, LNCS. Springer, 2000.
6. B. Brumley and N. Tuveri. Remote timing attacks are still practical. *ESORICS*, 2011.
7. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0, 2000.

8. M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Designs, Codes and Cryptography*, 2005.
9. P. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 1990.
10. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *CHES*, LNCS. Springer, 1999.
11. N. Ebeid and R. Lambert. Securing the Elliptic Curve Montgomery Ladder Against Fault Attacks. In *FDTC*, pages 46–50. IEEE Computer Society, 2009.
12. J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-Art of Secure ECC Implementations: A Survey on known Side-Channel Attacks and Countermeasures. In *HOST*. IEEE, 2010.
13. J. Fan and I. Verbauwhede. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In *Cryptography and Security: From Theory to Applications*, LNCS. Springer, 2012.
14. P.-A. Fouque, R. Lercier, D. Réal, and F. Valette. Fault Attack onElliptic Curve Montgomery Ladder Implementation. In *FDTC*. IEEE Computer Society, 2008.
15. Freescale Semiconductor. Kinetis L Series MCUs, 2013. Available online at `http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=KINETIS_L_SERIES`.
16. Fujitsu Semiconductors. Fujitsu Semiconductor Widely Expands Lineup of 32-bit General Purpose Microcontrollers with the Release of Products Adopting 2 New ARM Cores, November 2012. Press Release.
17. F. Fürbass and J. Wolkerstorfer. ECC Processor with Low Die Size for RFID Applications. In *IEEE International Symposium on Circuits and Systems*, 2007.
18. L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *PKC*, LNCS, 2003.
19. C. P. L. Gouvêa and J. López. Software Implementation of Pairing-Based Cryptography on Sensor Networks Using the MSP430 Microcontroller. In *INDOCRYPT*, 2009.
20. C. P. L. Gouvêa, L. Oliveira, and J. López. Efficient Software Implementation of Public-Key Cryptography on Sensor Networks Using the MSP430X Microcontroller. *Journal of Cryptographic Engineering*, 2012.
21. J. Großschädl and E. Savaş. Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In *CHES*, 2004.
22. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs. In *CHES*, LNCS. Springer, 2004.
23. J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, and G. Sigl. Localized Electromagnetic Analysis of Cryptographic Implementations. *CT-RSA*, 2012.
24. M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In *RFIDSec*, 2010.
25. M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In *AFRICACRYPT*, 2011.
26. M. Hutter and E. Wenger. Fast Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors. In *CHES*, LNCS. Springer, 2011.
27. T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronic Letters*, 1988.
28. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In *CHES*, LNCS, 2003.
29. T. Kern and M. Feldhofer. Low-Resource ECDSA Implementation for Passive RFID Tags. In *ICECS*, pages 1236–1239. IEEE, 2010.

30. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*, LNCS. Springer, 1996.
31. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, LNCS. Springer, 1996.
32. A. K. Lenstra, H. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 1982.
33. A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *International Conference on Information Processing in Sensor Networks*, 2008.
34. P. L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 1987.
35. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009.
36. NXP Semiconductors. NXP Licenses ARM Cortex-M0+ Processor, March 2012. Press Release.
37. Olivier Girard. openMSP430, 2013. Available online at `http://opencores.org/project,openmsp430`.
38. E. Öztürk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In *CHES*, LNCS, pages 92–106, 2004.
39. A. Satoh and K. Takano. A Scalable Dual-Field Elliptic Curve Cryptographic Processor. *IEEE Transactions on Computers*, 2003.
40. J.-M. Schmidt and C. Herbst. A Practical Fault Attack on Square and Multiply. In *FDTC*. IEEE Computer Society, 2008.
41. J.-M. Schmidt and M. Medwed. A Fault Attack on ECDSA. In *FDTC*. IEEE, 2009.
42. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In *Wireless Sensor Networks 5th European Conference*, LNCS. Springer, 2008.
43. Texas Instruments. MSP430 Ultra-Low Power 16-Bit Microcontrollers, 2013. Available online at `http://www.ti.com/msp430`.
44. T. Unterluggauer. Xetroc-M0+. An implementation of ARMs Cortex-M0+. Master project, Graz University of Technology, 2013.
45. H. Wang, B. Sheng, and Q. Li. Elliptic Curve Cryptography-based Access Control in Sensor Networks. *International Journal of Security and Networks*, 2006.
46. E. Wenger. A Lightweight ATmega-based Application-Specific Instruction-Set Processor for Elliptic Curve Cryptography. In *LightSec*, LNCS, pages 1–15, 2013.
47. E. Wenger, T. Baier, and J. Feichtner. JAAVR: Introducing the Next Generation of Security-Enabled RFID Tags. In *DSD*, pages 640–647. IEEE, 2012.
48. E. Wenger, M. Feldhofer, and N. Felber. Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In *WISA*, 2010.
49. E. Wenger and M. Werner. Evaluating 16-bit Processors for Elliptic Curve Cryptography. In *Smart Card Research and Advanced Applications*. Springer, 2011.
50. M. Werner. IDLE430 - an ImproveD msp LikE processor. Master project, Graz University of Technology, 2013.
51. S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. In *IEEE Transactions on Computers*. IEEE, 2000.

A Analysis of Point Multiplication Formula

The following analysis discusses how Algorithm 1 holds up against the most common side-channel attacks. It is based on the overview papers of Fan *et al.* [12, 13]. Attacks that are considered not to affect the security of the given algorithm:

Timing analysis [30] is not possible, because the used Montgomery Ladder [25] has a key-independent runtime, and all finite-field operations have a constant runtime as well. To avoid leading-zero-bits timing attacks [6] based on the LLL algorithm [32], we set the most-significant bit of the secret ephemeral scalar to one.

Simple power analysis [31] is hindered by using a Montgomery Ladder and Randomized Projective Coordinates.

Differential power analysis [31], Refined power analysis [18] are not possible as random ephemeral scalars are used for DHKE and ECDSA.

M & C safe-error analysis [28, 51] are not possible because a Montgomery ladder in conjunction with random ephemeral scalars is used.

Invalid point analysis [5], Twist-curve based analysis [14] is not possible because in lines 1, 3, 10, and 12 point-validity checks are performed.

Subgroup analysis [5] is not possible because an y-recovery with subsequent point verification is performed. Ebeid and Lambert [11] provide a thorough analysis of the y-recovery as countermeasure.

Attacks that may affect the security of the given algorithm:

Program-flow fault analysis [40, 41]. A fault attack applied on the program flow can hardly be detected by the program flow itself. To circumvent such paths of attacks, hardware countermeasures are recommended.

Invalid-curve analysis [8] has to be additionally handled by checking the validity of the stored curve parameters. This is not done within Algorithm 1, but the point-validity checks certainly handicap any invalid-curve attack.

Electromagnetic-emanation analysis [23] is possible if the attacker can detect which memory locations are accessed at certain points in time. A countermeasure would be to randomize the memory access patterns.