Efficient Pairings and ECC for Embedded Systems

Thomas Unterluggauer and Erich Wenger

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, 8010 Graz, Austria
{Thomas.Unterluggauer,Erich.Wenger}@iaik.tugraz.at

Abstract. The research on pairing-based cryptography brought forth a wide range of protocols interesting for future embedded applications. One significant obstacle for the widespread deployment of pairing-based cryptography are its tremendous hardware and software requirements. In this paper we present three side-channel protected hardware/software designs for pairing-based cryptography yet small and practically fast: our plain ARM Cortex-M0+-based design computes a pairing in less than one second. The utilization of a multiply-accumulate instruction-set extension or a light-weight drop-in hardware accelerator that is placed between CPU and data memory improves runtime up to six times. With a 10.1 kGE large drop-in module and a 49 kGE large platform, our design is one of the smallest pairing designs available. Its very practical runtime of 162 ms for one pairing on a 254-bit BN curve and its reusability for other elliptic-curve based crypto systems offer a great solution for every microprocessor-based embedded application.

Keywords: optimal-ate pairing, elliptic-curve cryptography, embedded computing, hardware/software co-design.

1 Introduction

The field of pairing-based cryptography has become the key enabler for novel protocols and algorithms: privacy-aware group-signature schemes [9, 22], identity-based encryption schemes [7, 23], and since recently even provable leakage-resilient protocols [25] rely on pairing operations. The practical advantages of those protocols motivate their use in the very competitive markets of embedded microprocessors and smart cards.

The biggest implementation challenges of pairing-based cryptography are related to its tremendous resource and runtime requirements. Therefore, researchers started to implement optimized pairing operations for desktop computers [1, 6], for smart phones [20, 31], and as dedicated hardware modules [16, 24].

Cost-sensitive embedded applications however simply do not have the budget for such powerful application processors or 130-180 kGE of dedicated hardware.

For these embedded scenarios, implementations on light-weight RISC processors have been done. For example, Szczechowiak *et al.* [33] need 17.9 seconds for a pairing on an ATmega microprocessor, Gouvêa *et al.* [18] need 1.9 seconds on an MSP430X microprocessor, and Devegili *et al.* [15] need 2.5 seconds on a Philips HiPerSmart™ MIPS microprocessor. Unfortunately, such runtimes are not very promising for real-world, interactive applications as pairing-based protocols like group-signature schemes often happen to rely on several pairing and group operations. The resulting overall runtimes of several seconds would be considerably too slow. Additionally, it is unclear to which degree timing-analysis, power-analysis, or fault-analysis attacks have been considered in all those implementations.

These limitations motivated us to be the first to implement constant-runtime, side-channel protected optimal-Ate pairings using Barreto-Naehrig (BN) curves [4] on an ARM Cortex-M0+ [2, 3] microprocessor[1]. The respective pairing runtime of 993 ms seems very promising as it is several times faster than related work[2], but might be insufficient for interactive protocols as well. Therefore, it was a necessity to improve performance by adding dedicated hardware.

In this paper, we present three reusable pairing platforms which offer runtimes of down to 162 ms requiring 10.1 kGE of dedicated hardware at most – significantly less than similarly fast hardware implementations by related work. Our rigorous hardware/software co-design approach equipped one platform with a multiply-accumulate instruction-set extension and another platform with a drop-in accelerator[3] [35]. By building a flexible, specially crafted drop-in module with several novel design ideas, we were able to improve the runtime of pairing and group operations up to ten times. This concept platform consisting of CPU, RAM, ROM, and drop-in module consumes merely 49 kGE of hardware in total with 10.1 kGE of those being spent for the drop-in accelerator. The practicability of this platform is evaluated for several high-level pairing protocols [7, 8, 22] – each operating in significantly less than one second. Its reusability for Elliptic-Curve Cryptography (ECC) is further verified for `secp160r1`, `secp256r1` [11, 29], and Curve25519 [5], requiring 11.9-36.8 ms for a side-channel protected point multiplication. Those results make the drop-in based platform highly suitable for embedded computing, smart cards, wireless sensor nodes, near-field communication, and the Internet of Things.

The paper is structured as follows: Section 2 gives an overview on pairings and Section 3 covers the implementation aspects of the high-level pairing arithmetic. In Section 4, the architectural options to build suitable pairing platforms are presented. The respective platforms are evaluated in Section 5 and compared

[1]The source code is available at `https://github.com/IAIK/pairings_in_c`.

[2]Not considering the different underlying microprocessor architectures.

[3]Wenger [35] applied the concept to binary-field based elliptic-curve cryptography while we apply the concept to prime-field based elliptic-curve cryptography.

with related work in Section 6. The (re-)usability of our drop-in platform is content of Section 7. A conclusion is finally done in Section 8.

2 Background on Pairings

The wide range of cryptographic protocols in pairing-based cryptography is based on three cyclic order-n groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and a bilinear pairing operation. A bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ accepts an element of the two additive groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, maps these to the multiplicative group $\mathbb{G}_T$, and hereby fulfills several properties:

1. Bilinearity: $e(aP, bQ) = e(P, Q)^{ab} \;\; \forall P \in \mathbb{G}_1,\; Q \in \mathbb{G}_2,\; a, b \in \mathbb{Z}$.
2. Non-degeneracy: $\forall P \in \mathbb{G}_1 \setminus \{\mathcal{O}\} \; \exists \, Q \in \mathbb{G}_2 : e(P, Q) \neq 1$.
3. Computability: $e(P, Q)$ can be computed efficiently.

The groups $\mathbb{G}_1$, $\mathbb{G}_2$ are typically groups over elliptic curves and $\mathbb{G}_T$ is the subgroup of a large extension field. However, only certain elliptic curves allow the definition of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ with an admissible bilinear pairing, e.g., [4, 27]. In this paper, we focus on the pairing-friendly elliptic curves by Barreto and Naehrig [4] of the form $E : y^2 = x^3 + b$ with $b \neq 0$ (*BN curves*). Ate pairings $a(Q, P)$ based on these curves can be described as follows:

$$a\colon \; \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T \; : \; E(\mathbb{F}_{p^{12}}) \times E(\mathbb{F}_p) \to \mathbb{F}^*_{p^{12}} \, . \qquad (1)$$

Note that for $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ to have the same prime order n, $\mathbb{G}_2$ and $\mathbb{G}_T$ need to be subgroups of $E(\mathbb{F}_{p^{12}})$ and $\mathbb{F}^*_{p^{12}}$, respectively. The BN curves use a parameter u such that a desired security level is achieved. This allows the computation of the prime p and the prime group order n in dependence of u:

$$\begin{aligned} p(u) &= 36u^4 + 36u^3 + 24u^2 + 6u + 1 \\ n(u) &= 36u^4 + 36u^3 + 18u^2 + 6u + 1 \, . \end{aligned}$$

As another benefit, BN curves possess an efficiently computable group homomorphism that exploits the curve's sextic twist E'. Utilization of this homomorphism allows the compression of the elements in $\mathbb{G}_2$, which leads to a more efficient definition of the Ate pairing, namely

$$a\colon \; \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T \; : \; E'(\mathbb{F}_{p^2}) \times E(\mathbb{F}_p) \to \mathbb{F}^*_{p^{12}} \, . \qquad (2)$$

The pairing a itself consists of the evaluation of a rational function $f_{\lambda,Q}$ and a final exponentiation that maps all cosets to the same unique representative:

$$a = f_{\lambda,Q}(P)^{(p^{12}-1)/n}.$$

Owing to the Frobenius homomorphism, the final exponentiation by $(p^{12} - 1)/n$ can be split into an easy part $(p^6 - 1)(p^2 + 1)$ and a hard part $(p^4 - p^2 + 1)/n$.

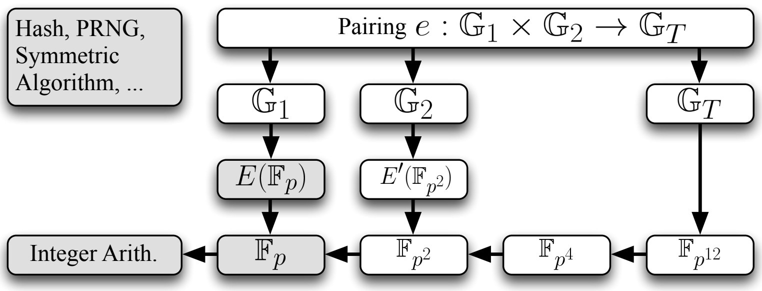


Fig. 1: Arithmetic required for pairings over Barreto-Naehrig curves

The function $f_{\lambda,Q}$ can in general not be evaluated directly. However, Miller [26] described an important property of rational functions, namely

$$f_{i+j,P} = f_{i,P} f_{j,P} \frac{\ell_{[i]P,[j]P}}{\nu_{[i+j]P}} .$$

The property allows the computation of $f_{\lambda,Q}$ in polynomial time by merely evaluating vertical (ν) and straight (ℓ) lines in elliptic curve points using a double-and-add approach. Values of λ with low Hamming weight result in a particularly fast computation of $f_{\lambda,Q}$, the pairing becomes optimal. In this work, we used the efficient optimal-Ate pairing by Vercauteren [34].

3 High-Level Arithmetic

The computation of bilinear pairings over BN curves requires several layers of arithmetic. As illustrated in Figure 1, all arithmetic is based on a multi-precision integer arithmetic layer. On top of that, prime-field arithmetic and a tower of extension fields are built upon. The elliptic curve groups used as $\mathbb{G}_1$ and $\mathbb{G}_2$ utilize the prime field and its quadratic extension field, respectively. The largest extension field $\mathbb{F}_{p^{12}}$ is used by $\mathbb{G}_T$. The pairing computation itself is based on the groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, and their underlying field arithmetic.

Methodology. Our state-of-the-art implementations are based on the techniques used by Beuchat *et al.* [6] and Devegili *et al.* [14]. The pairing implementation uses the fast formulas by Costello *et al.* [13], the inversion trick by Aranha *et al.* [1], a lazy reduction technique in $\mathbb{F}_{p^2}$ [6, 31], and a slightly modified variant of the final exponentiation by Fuentes-Castañeda *et al.* [17] that requires less memory (see Appendix A.1). The prime-field inversion using Fermat's little theorem is optimized according to Appendix A.2. Since operations in $\mathbb{G}_T$ and in the hard part of the final exponentiation take place in the cyclotomic subgroup of $\mathbb{F}^*_{p^{12}}$, dedicated squaring formulas are utilized [19]. The point multiplications in both elliptic curve groups use Montgomery ladders that are based on fast formulas [21] in homogeneous projective co-Z coordinates.

Parameters. As this work aims to offer a certain degree of flexibility, both the 80-bit and the 128-bit security level are supported. The two elliptic curves BN158 [18] ($u = \mathtt{40\,00800023}_h$) and BN254 [30] ($u = -\mathtt{40800000\,00000001}_h$) of the form $y^2 = x^3 + 2$ were chosen. Those lead to particularly fast execution times as the respective constants λ of $f_{\lambda,Q}$ have low Hamming weights. The extension field $\mathbb{F}_{p^2}$ is represented as $\mathbb{F}_p[i]/(i^2 - \beta)$ with $\beta = -1$. The extension field $\mathbb{F}_{p^{12}}$ is built as $\mathbb{F}_{p^2}[z]/(z^6 - \zeta)$, with $\zeta = (1 + i)$ for BN254 and $\zeta = \frac{1}{1+i}$ for BN158.

Implementation Attacks. An important aspect in the implementation of pairings and group arithmetic for embedded applications is the consideration of side-channel attacks. While scalar factors or exponents are typically the secret operands for operations in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, an elliptic curve point may have to be protected in the case of pairing operations.

As a countermeasure to timing attacks, all implemented algorithms have constant, data-independent runtime. Therefore, e.g., some fast but vulnerable point multiplication algorithms are not used. Both the point multiplications in $\mathbb{G}_1$, $\mathbb{G}_2$ and the exponentiations in $\mathbb{G}_T$ hence use Montgomery ladders. The implementation's countermeasures against first-order Differential Power Analysis (DPA) attacks comprise Randomized Projective Coordinates (RPC) [12] in both the pairing computation and the point multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$. To detect fault attacks on data, point multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$ include several point verifications. DPA and fault attacks on exponentiations in $\mathbb{G}_T$ as well as fault attacks on pairings were also taken into consideration, but can better be handled on the protocol layer using randomization.

4 Hardware Architectures

To meet the high requirements of pairing-based cryptography in embedded devices, our goal was to equip a stand-alone microprocessor, designated for embedded applications, with a dedicated hardware unit such that: (i) Pairing computations are usable within interactive (e.g., authentication) protocols. (ii) A pre-existing microprocessor platform is modified only minimally. (iii) The overall hardware requirements, i.e., the costs, are kept small and considerably below 100 kGE needed in related work [16, 24]. (iv) Embedded applications such as wireless sensor nodes and NFC should be practically feasible.

Figure 2 summarizes potential architectures that can be used to attain such goals. The straightforward solution *(a)*, a sole off-the-shelf microprocessor, requires minimal hardware-development time, however potentially delivers insufficient performance. The runtimes desirable for interactive protocols can only be achieved by either adding powerful, dedicated instructions *(b)*, or by adding dedicated co-processors. Contrary to a dedicated hardware module *(c)*, a drop-in module *(d)* is memoryless and requires neither a Direct Memory Access (DMA) controller nor a multi-master bus. Wenger [35] showed the advantages of the drop-in concept in comparison to a dedicated hardware module for binary-field ECC. However, the applicability of this technique for prime-field based pairings is still an open question.

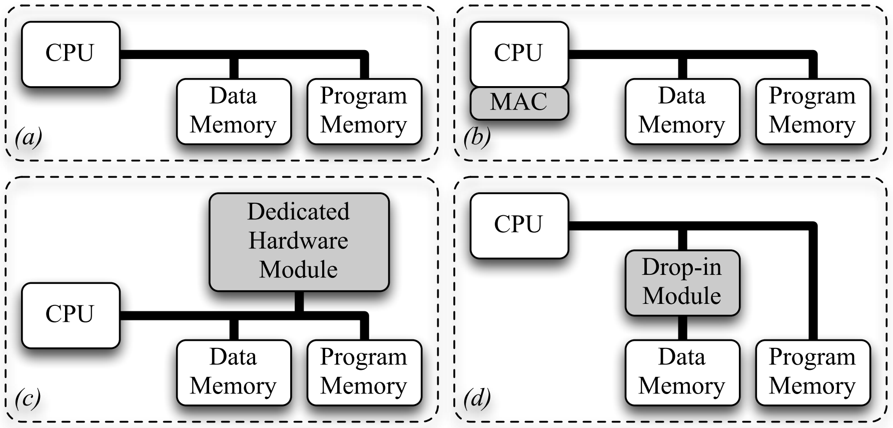


Fig. 2: Architectural options for fast and flexible pairing designs

Following up the potential architectures, we consecutively evaluate the practicability of a plain microprocessor design *(a)*, a multiply-accumulate instruction-set extension *(b)*, and a dedicated drop-in module *(d)*.

4.1 The Used Microprocessor

The accomplishment of the initially set goals highly depends on the used microprocessor. As the runtime figures by Szczechowiak *et al.* [32] and Gouvêa *et al.* [18] discourage the use of an 8-bit or 16-bit microprocessor, a 32-bit microprocessor is preferred as a basis. Moreover, the bottleneck between computation unit and RAM is less of an issue if 32-bit interfaces are used. We hence decided to utilize a self-built processor functionally equivalent to the ARM Cortex-M0+ [2], because the Cortex-M0+ was especially designed for embedded applications and currently is one of the smallest 32-bit processors in production. The Cortex-M0+ has 16 32-bit general-purpose registers of which 8 are efficiently usable. It comes with a mixed 16/32-bit Thumb/Thumb-2 instruction set and optionally either a 32-cycle or single-cycle 32-bit multiplier. In its minimum configuration, ARM specifies its Cortex-M0+ to require only 12 kGE in a 90 nm process technology.

4.2 The Software Framework

The biggest advantage of an off-the-shelf microprocessor are the vast (open-source) toolchains. Thus a high-level framework capable of pairing-based cryptography using BN curves was created in C. It provides extension field arithmetic, elliptic curve operations, and bilinear pairings. The framework focuses on both good performance and low memory consumption. To achieve the latter, several optimizations were incorporated into the framework. First, virtually all of the memory is allocated on the stack. As stack variables are discarded at the end of each function, stack allocation facilitates the reduction of required memory by separating code into different functions. Second, allocated memory is reutilized where possible. Third, memory-optimized algorithms are used, e.g., for the final exponentiation as in Appendix A.1. Last, compiler optimizations are used to decrease the program size. Therefore, the compiler options `-ffunction-sections`, `-fdata-sections` and the linker options `-gc-sections`, `--specs=nano.specs` are passed to the bare-metal ARM GNU toolchain (version 4.7.4).

The high-level pairing framework is common to all three evaluated platforms. The main difference between these platforms is the implemented finite-field arithmetic. While *(a)* and *(b)* control the whole finite field arithmetic in software, *(d)* relies on finite-state machines to perform additions, subtractions and multiplications in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. Nevertheless, all implementation options ensure constant runtime and consider side-channel attacks.

4.3 Assembly-Optimized Software Implementation *(a)*

The plain microprocessor platform *(a)* is based on a Cortex-M0+ with a single-cycle multiplier. Its hand-crafted assembly routines for optimized prime-field arithmetic always perform a reduction step to ensure constant runtime. This is accomplished by storing the reduction result either to the target or a dummy memory location via masking of the operand addresses. The crucial prime-field multiplication utilizes an unrolled Separated Product Scanning (SPS) method of the Montgomery multiplication [28] that is derived from [10]. The SPS variant is chosen because of the particular $\mathbb{F}_{p^2}$-multiplication technique [6, 31] we use, which performs the required three multiplications and two reductions separately. Product scanning can further be efficiently implemented on the processor if three registers are used as an accumulator, as presented in [36]. The reduction step for the curve BN254 is further optimized as several multiply-accumulates can be skipped due to the sparse prime [18].

4.4 Multiply-Accumulate Hardware Extensions *(b)*

The performance of the prime-field multiplication significantly suffers from the $32 \times 32 \rightarrow 32$ bit multiplier of the Cortex-M0+, which results in 80% of a pairing's runtime being spent in $\mathbb{F}_p$ multiplications. To improve this, the processor core is equipped in *(b)* with a multiply-accumulate extension similar to [36]. It adds the result of a full $32 \times 32 \rightarrow 64$ bit multiplication to three accumulation registers in a single cycle. In order to avoid a modification of the compiler toolchain, the `TST` instruction, which is not required for prime-field multiplication, is reinterpreted as a multiply-accumulate instruction if a certain bit in the control register is set. The control register is manipulated accordingly at the beginning and the end of a prime-field multiplication. Besides accelerated multiply-accumulate operations, the prime-field multiplication requires less registers for temporary variables, which we exploit by caching some of the operand words in the product scanning routine.

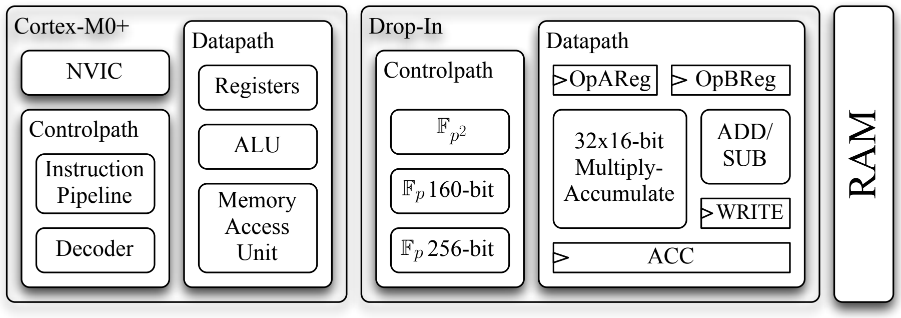


Fig. 3: High-level representation of architecture *(d)* (without program memory). Note that the sizes of the blocks are not proportional to their respective hardware footprints

4.5 The Drop-in Module *(d)*

As a consequence of the high-level runtime and area goals, it is of utmost importance to maximize the utilization of the invested chip hardware. To achieve this, a lightweight hardware drop-in accelerator is placed between processor and data memory. The respective design, which is shown in Figure 3, uses a Cortex-M0+, but any other processor is equally suitable.

The drop-in module provides unrolled state machines and an appropriate arithmetic unit for 160-bit and 256-bit $\mathbb{F}_p$ multiplication, $\mathbb{F}_p$ addition and $\mathbb{F}_p$ subtraction. It further encompasses state machines to control $\mathbb{F}_{p^2}$ addition, $\mathbb{F}_{p^2}$ subtraction, $\mathbb{F}_{p^2}$ multiplication and $\mathbb{F}_{p^2}$ squaring. Several memory-mapped registers are used to control the drop-in module. A lightweight arbiter is built in which always gives preference to the CPU when the CPU wants to access the data memory. In such case, the drop-in module is prepared to stall its operation.

The core element of our drop-in module is a multiply-accumulate unit that is used to perform a Finely Integrated Product Scanning (FIPS) [10] Montgomery multiplication. Within this algorithm approximately $2N^2 + N$, with $N = \lceil \frac{\mathrm{ld}(p)}{W} \rceil$, W-bit integer multiplications are performed that require approximately $4N^2$ load operations. Instead of using a dual-port memory, we attain a perfectly utilized bus and a perfectly utilized multiplier by using a two-cycle multiply-accumulate unit that is based on a $W \times W/2$-bit multiplier. This saves 3 kGE for $W = 32$ in an 130 nm process compared to a traditional $W \times W$-bit multiplier.

A finite-field operation is started by writing three memory pointer registers (`OpA`, `OpB`, and `RES`) and a control register. As those registers are mapped at consecutive addresses, the store-multiple instruction (`STM`) of the Cortex-M0+ can be used to efficiently start an operation. A started finite-field multiplication is performed using the following hardware components: a $W \times W/2 = 32 \times 16$-bit multiplier, a $\lceil \mathrm{ld}(2N) \rceil + 2W = 68$-bit `ACC`umulator, a $W = 32$-bit register for operand A (`OpAReg`), a $3W/2 = 48$-bit register for operand B (`OpBReg`), and a $W = 32$-bit `WRITE` register. In `OpBReg`, the top 32 bits are always written by the bus and the lowest 16 bits are used as an operand of the multiplier. Therefore, a

Table 1: Propagation of data within the pipelined drop-in module

Bus	`OpBReg`	`OpAReg`	**Mult.**	**Accum.**
`LD OpB+0`				
`LD OpA+0`	`WR`			
`LD OpB+0`	`SH`	`WR`		
`LD OpA+1`	`WRSH`		`MUL1`	
`LD OpB+1`	`SH`	`WR`	`MUL2`	`SHIFT`
`LD OpA+0`	`WRSH`		`MUL1`	
`LD OpB+2`	`SH`	`WR`	`MUL2`	
`ST RES+0`	`WRSH`		`MUL1`	
`LD OpB+1`	`SH`		`MUL2`	`SHIFT`
`LD OpA+1`	`WRSH`		`MUL1`	
`LD OpB+0`	`SH`	`WR`	`MUL2`	

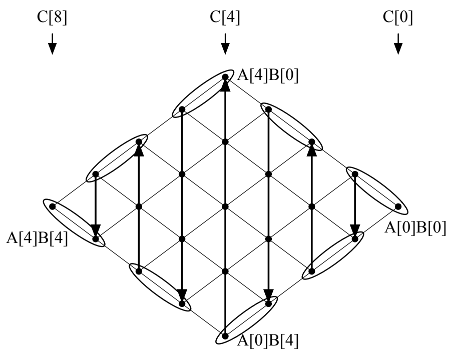


Fig. 4: 5×5-word zig-zag product scanning multi-precision multiplication method

sequence of shift/rotate operations is necessary to actually multiply the loaded operands. Table 1 visualizes the dataflow within the drop-in module. For a single multiply-accumulate operation five clock cycles are necessary. As the drop-in module heavily relies on pipelining, practically only two cycles are needed. The following steps are performed: (i) `OpB+i` is applied to the bus. (ii) `OpB+i` is `WR`itten to `OpBReg` and `OpA+j` is applied to the bus. (iii) `OpAReg` is `WR`itten and `OpBReg` is `SH`ifted by 16 bits. (iv) The first multiplication cycle (`MUL1`) multiplies the lower 16 bits of `OpB+i` with `OpA+j` and `OpBReg` is shifted again. (v) During the second multiplication cycle (`MUL2`) the accumulator is optionally `SHIFT`ed. When shifted, the lowest 32-bit of the accumulator are stored in the `WRITE` register. This data is later written to the address `RES+i+j`, when the bus is not utilized.

As the fully utilized bus needs some free cycles to write the result, we use a zig-zag product scanning technique (cf. Figure 4) [37]. In this technique, consecutive columns are traversed in different order, which allows caching of a single operand from one column to the next. This frees the bus for $2N$ cycles, which are exactly the $2N$ cycles required to store the computed results.

Although the implemented FIPS multiplication is quite complex, the software running on the CPU is completely independent of the methodology used to perform finite-field arithmetic within the drop-in module. However, there are two implementation guidelines the software has to deal with. First, constant variables have to be temporarily copied to the data memory when being used. Second, there are two techniques to wait for the drop-in module to finish. A function delegating an operation to the drop-in module can either start an operation and wait for it to finish, or wait for a previously started operation to finish and only then start a new operation. The latter case is more performant because the CPU and the drop-in module potentially work in parallel, i.e., the control flow operations involved in the invocation of the routines that call the drop-in module are done while the drop-in module is computing. However, temporary variables

Table 2: Performance of various operations on architectures *(a)*, *(b)*, and *(d)*

Design	$\mathbb{F}_\mathbf{p}$ **Add** [Cycles]	**Mul** [Cycles]	**Inv** [kCycles]	$\mathbb{G}_\mathbf{1}$ **Mul** [kCycles]	$\mathbb{G}_\mathbf{2}$ **Mul** [kCycles]	$\mathbb{G}_\mathbf{T}$ **Exp** [kCycles]	$\mathbb{G}_\mathbf{1} \times \mathbb{G}_\mathbf{2}$ **Pairing** [kCycles]	**RAM** [Byte]	**ROM** [Byte]
BN158									
Cortex-M0+	112	1,800	331	4,828	11,775	22,871	17,389	1,856	13,980
MAC	112	361	72	1,129	4,042	10,736	7,828	1,796	11,232
Drop-in	56	161	29	493	1,577	4,322	3,182	1,876	10,364
BN254									
Cortex-M0+	166	3,782	1,122	16,071	38,277	72,459	47,643	2,828	18,116
MAC	166	934	285	4,323	11,449	27,460	17,960	2,836	12,572
Drop-in	75	335	97	1,566	4,858	12,076	7,763	2,880	10,764

on the stack are freed once a function finishes, which requires adding additional wait statements within the extension-field arithmetic to prevent the drop-in from accessing reallocated memory locations. Nevertheless, the utilization of the drop-in is increased from 77.6% to 85.1% when the function first waits for previous operations to finish. Similarly, the utilization of the RAM is raised from 75.7% to 80.1% (cf. 34.6% in *(b)*, 17.0% in *(a)*).

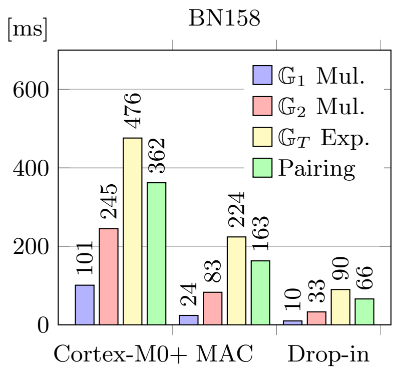
5 Implementation Results

To verify the achievement of the area and performance goals initially set, the three microprocessor-based platforms *(a)*, *(b)* and *(d)* were evaluated with respect to hard- and software. Regarding the overall hardware platforms, runtime, area, power, and energy consumption are distinctive. Regarding the software part, the evaluation focuses on the runtimes of the underlying finite-field arithmetic and the most expensive operations used within protocols: the point multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$, the exponentiation in $\mathbb{G}_T$, and the pairing operation.

The results in Table 2 show that the multiply-accumulate extension speeds up the prime-field multiplications by factors of 4.0-5.0[4], but leaves the prime-field additions unaffected. The same speed-ups are observed for prime-field inversions and point multiplications in $\mathbb{G}_1$. However, the impact of the multiply-accumulate extension on the performance of both pairings and operations in $\mathbb{G}_2$, $\mathbb{G}_T$ is lower and lies between a factor of 2.1 and 3.3. Considering the performance of the drop-in module, an even greater speed-up is observed compared to the plain software implementation. In this case, prime-field multiplications, inversions and point multiplications in $\mathbb{G}_1$ are up to 11.3 times faster, which eventually results in an up to 6.1 times faster computation of pairings. On average, operations using BN158 are 3.0 times faster than operations using BN254.

Throughout all implementations, the demand for data memory is kept relatively low, with a maximum of 1,876 bytes and 2,880 bytes for BN158 and BN254, respectively. Similarly, the program sizes are kept small, e.g., 18 KB for BN254.

[4]The implementation for BN158 with multiply-accumulate extension utilizes the FIPS method and discards lazy reduction in $\mathbb{F}_{p^2}$ [6, 31] as it yields better performance.



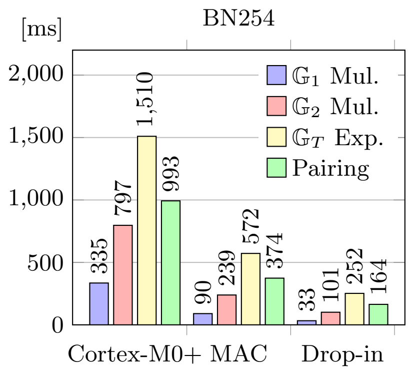


Fig. 5: Group operations at 48 MHz

Given a typical clock frequency of 48 Mhz, the performance results of the point multiplications in $\mathbb{G}_1$, $\mathbb{G}_2$, the exponentiation in $\mathbb{G}_T$, and the pairing operation are illustrated in Figure 5. The respective runtimes support our choice of a 32-bit architecture: providing 128-bit security, the drop-in based platform does pairing computations in highly practical 164 ms. The pure *embedded* software implementation performs the same computation in 993 ms.

While Table 2 focuses on the software part, the most important hardware characteristics are visualized in Table 3. The runtime is given for a single pairing computation. Both area and power measurements were determined for an 130 nm low-leakage UMC technology. The area results in a 90 nm UMC technology are explicitly marked. The designs were synthesized and their power and runtime evaluated for a clock frequency of 48 MHz. Both data and program memory were realized using RAM and ROM macros of appropriate sizes. The program memory encompasses all routines required to implement pairing-based protocols, i.e., pairings, operations in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. These platforms are hence ready-to-use for future applications based on pairings over BN curves.

According to Table 3, BN254 pairing computations with reasonable performance are available at the cost of 57.7 kGE in an 130 nm process technology. Switching to the more advanced 90 nm process technology shrinks the design to 49.0 kGE, constituting one of the smallest available hardware designs for pairings with practical relevance. In terms of power consumption, the plain microprocessor design is, as expected, the most economical. The multiply-accumulate extension and the drop-in module increase power consumption by 25% and 70%, respectively. Due to their increased performance, these platforms are more energy-efficient though. Their respective demand for energy is 2.1 and 3.5 times lower.

Table 3: Implementation characteristics for 130 nm and 90 nm process technologies

Platform	Area RAM [kGE]	ROM [kGE]	CPU [kGE]	Dedicated [kGE]	Total [kGE]	Power [mW]	Runtime [ms]	Energy [mJ]
BN158								
Cortex-M0+	11.4	15.6	18.4	-	45.4	5.92	362	2.14
MAC	11.1	13.8	27.1	-	52.0	7.38	163	1.20
Drop-in	11.4	13.8	17.0[a]	10.8	52.9	10.25	66	**0.68**
Drop-in 90nm	10.5	12.0	12.6[a]	**10.1**	**45.2**	-	66	-
BN254								
Cortex-M0+	16.0	19.3	18.4	-	53.7	5.80	993	5.76
MAC	16.0	15.6	27.1	-	58.8	7.33	374	2.74
Drop-in	16.2	13.8	17.0[a]	10.8	57.7	9.96	162	**1.61**
Drop-in 90nm	14.3	12.0	12.6[a]	**10.1**	**49.0**	-	162	-

[a]Bit-serial multiplier.

6 Comparison with Related Work

As a consequence of our hardware/software co-design approach, comparison with related work focuses on two aspects. On the one hand, the pure software implementation on the Cortex-M0+ is brought into relation to other software implementations on low-resource hardware. On the other hand, the resulting hardware design is compared with other dedicated pairing hardware implementations.

The comparison of our software implementation with related implementations of Ate pairings over BN curves providing approximately 128-bit security is summarized in Table 4. Gouvêa *et al.* [18] provide highly optimized software implementations for the 16-bit microcontroller MSP430 and a variant of its successor MSP430X, which is equipped with a 32-bit multiplier (MPY32). The implementation by Devegili *et al.* [15] is evaluated on a 32-bit Philips HiPerSmart™ smart card, which has a SmartMIPS architecture and clearly is a direct competitor of Cortex-M0+-based smart cards. However, it is unclear to which extent side-channel resistance is considered by either of them.

As both the MSP430 and the Cortex-M0+ use a 16-bit instruction-set, it is important to highlight the exceptionally low program and data memory footprint of our implementations. It is however hard to compare the quality of an

Table 4: Related software implementations of Ate pairings over BN curves

	Platform	RAM [Byte]	ROM [Byte]	Runtime [kCycles]	Frequ. [MHz]	Runtime [ms]
Gouvêa [18]	MSP430	6,500	36,000	79,440	8	9,930
Devegili [15]	Philips HiPerSmart™	<16,000	-	90,462	36	2,513
Gouvêa [18]	MSP430X/MPY32	6,500	34,400	47,736	25	1,909
Ours	Cortex-M0+	**2,828**	**18,116**	**47,643**	48	**993**

Table 5: Related hardware platforms (130 nm)

	Area		Time
	Ded. [kGE]	**Total** [kGE]	[kCycles]
Fan [16]	183	183	**593**
Kammler [24]	71[a]	164	5,340
Kammler [24]	67[a]	145	6,490
Kammler [24]	53[a]	130	10,816
Ours (Drop-in)	**11**[b]	**58**	7,763

[a]Core excl. 26 kGE of original RISC
[b]Drop-in module.

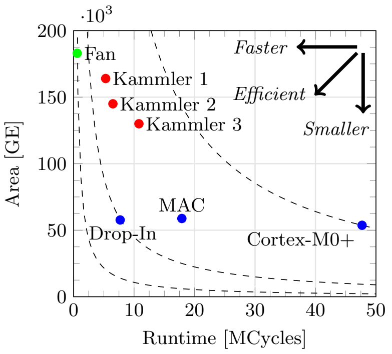


Fig. 6: Characteristics of related hardware

implementation when different frameworks and different microprocessors are involved.

Other pairing implementations for 32-bit ARM processors are limited to the Cortex-A series, such as in [20]. However, their pairing's runtime of 9.9 ms on a 1.2 GHz Cortex-A9 is as well hardly comparable with our pairing's runtime on the Cortex-M0+ since the multi-core Cortex-A processors provide massively higher clock frequencies along with a more powerful instruction set.

Regarding related hardware platforms, Table 5 covers hardware implementations of pairings providing roughly 128-bit security. Fan *et al.* [16] proposed a dedicated pairing cryptoprocessor with parallelized, full-precision $\mathbb{F}_p$ arithmetic. Its centerpiece is a hardware implementation of a hybrid modular multiplication algorithm that performs both polynomial and coefficient reduction. Their area figures, however, exclude the required RAM. Kammler *et al.* [24] extended a 5-stage 32-bit RISC core with instructions for $\mathbb{F}_p$ arithmetic. Their Application-Specific Instruction-set Processor (ASIP) uses a Montgomery multiplier structure that can be synthesized in different configurations and sizes. Unfortunately, their area figures do not contain the program memory.

In comparison to [16] and [24], our drop-in-based platform is 2.2-3.1 times smaller with regard to total area consumption. In both [24] and our case the CPU and the data memory can be reused for other applications. In terms of dedicated hardware, our drop-in-based platform is 16.6 times smaller than the work of Fan *et al.* In exchange, their design is faster and provides the best area-runtime product according to Figure 6. However, it depends on the application how much hardware area is actually acceptable to be spent on a dedicated pairing accelerator.

Table 6: Performance of pairing-based protocols on the drop-in platform

	$\mathbb{G}_1$ Mul	$\mathbb{G}_2$ Mul	$\mathbb{G}_T$ Exp	$\mathbb{G}_1 \times \mathbb{G}_2$ Pairing	**BN158** [ms]	**BN254** [ms]
Leakage Resilient KEM [25]						
Encaps.	0	1	1	0	123	353
Decaps.	2	0	0	2	153	389
Identity-Based Encryption KEM [7, 23]						
Encaps.	3	0	1	0	121	349
Decaps.	0	0	0	1.5[a]	99	243

[a]Ratios and products of pairings are counted as 1.5 pairing computations.

	$\mathbb{G}_1$ Mul	$\mathbb{G}_2$ Mul	$\mathbb{G}_T$ Exp	$\mathbb{G}_1 \times \mathbb{G}_2$ Pairing	**BN158** [ms]	**BN254** [ms]
Short Signatures [8]						
Sign	1	0	0	0	10	33
Verify	0	2	0	1	132	364
Short Group Signatures [22]						
Sign	9	2	0	1.5[a]	258	739
Verify	9	2	0	3	357	981
Link	0	0	0	3	199	485

7 Re-usability of our Drop-in Architecture

To emphasize the practicability of our low-area platforms for deploying cryptography to embedded environments, several protocols that are relevant in such context have been assessed in terms of the performance to expect.

Using the Drop-in Module for Pairing-based Protocols. The short signature scheme by Boneh *et al.* [8] is interesting for constrained signature devices as it aids to reduce communication. As a representative of group signatures, which help to provide anonymous authentication, the scheme by Hwang *et al.* [22] was chosen. To be able to establish a random session key without the necessity of verifying public keys, the identity-based encryption scheme by Boneh *et al.* [7] in its Key Encapsulation Mechanism (KEM) variant was evaluated as it combines good performance with small parameters. Additionally, the leakage resilient bilinear ElGamal KEM by Kiltz and Pietrzak [25] is taken into consideration because it is proven to have bounded side-channel leakage.

The number of computationally expensive operations and the expected overall runtime of each of the aforementioned protocols are presented in Table 6. The runtimes are given for the drop-in module based platform. As the figures suggest, all of the protocols may be performed on the device with user interaction as response times lie noticeably below one second.

Using the Drop-in Module for ECC. In order to emphasize the reusability of our drop-in module based design, we also evaluated the performance of the standardized curves [11, 29] `secp160r1` and `secp256r1` and the performance of Curve25519 by Bernstein [5], which many people fancy as replacement curve of standardized NIST curves. Again, we follow the point multiplication methodology from [36], which relies on Montgomery ladders, randomized projective coordinates and multiple point validation checks. All implementations have similar hardware footprints and require 4.1 kGE (500 bytes) for RAM, 6.2 kGE (3,200 bytes) for ROM, 10.1 kGE for the drop-in module, 12.6 kGE for the Cortex-M0+, and 33 kGE in total (in a 90 nm UMC technology). Point multiplications for `secp160r1`, `secp256r1`, and Curve25519 need 570 kcycles, 1,765 kcycles, and 1,110 kcycles, respectively. Note that we do not take advantage of the special form of the underlying primes. However, with runtimes of

11.9-36.8 ms (at 48 MHz) the drop-in concept is clearly an enabler of elliptic-curve based interactive protocols.

8 Conclusion

According to our evaluations of three microprocessor-based hardware designs, the utilization of a compact 32-bit microprocessor results in notably small pairing implementations. Requiring merely 45.2-49.0 kGE of chip area, we provided one of the smallest available hardware designs capable of bilinear pairings. The most prominent platform was however obtained by the construction of a dedicated drop-in hardware module for prime-field arithmetic. Its low area requirements and highly practical runtime facilitate pairing-based cryptography in interactive embedded applications.

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 FP7 program under project number 610436 (project MATTHEW).

References

1. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C., López, J.: Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In: Paterson, K. (ed.) EUROCRYPT 2011, LNCS, vol. 6632, pp. 48–68. Springer Berlin Heidelberg (2011)
2. ARM Ltd.: Cortex-M0+ Processor (Sep 2013), `http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php`
3. Atmel Corporation: Atmel SAM D20 ARM-based Microcontroller Datasheet (Dec 2013), `http://www.atmel.com/Images/Atmel-42129-SAM-D20_Summary.pdf`
4. Barreto, P.S., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer Berlin Heidelberg (2006)
5. Bernstein, D.: Curve25519: New Diffe-Hellman Speed Records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. vol. 3958, pp. 207–228 (2006)
6. Beuchat, J.L., González-Díaz, J., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010, LNCS, vol. 6487, pp. 21–39. Springer Berlin Heidelberg (2010)
7. Boneh, D., Boyen, X.: Secure Identity Based Encryption Without Random Oracles. In: Franklin, M. (ed.) CRYPTO 2004, LNCS, vol. 3152, pp. 443–459. Springer Berlin Heidelberg (2004)
8. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. Journal of Cryptology 21(2), 149–177 (2008)
9. Boneh, D., Boyen, X., Shacham, H.: Short Group Signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer Berlin Heidelberg (2004)

10. Ç.K. Koç, T. Acar and B.S. Kaliski, Jr.: Analyzing and Comparing Montgomery Multiplication Algorithms. IEEE Micro 16(3), 26–33 (June 1996)
11. Certicom Research: Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0 (September 2000), `http://www.secg.org/`
12. Coron, J.S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES'99. LNCS, vol. 1717, pp. 292–302. Springer (1999)
13. Costello, C., Lange, T., Naehrig, M.: Faster Pairing Computations on Curves with High-Degree Twists. In: Nguyen, P., Pointcheval, D. (eds.) PKC 2010, LNCS, vol. 6056, pp. 224–242. Springer Berlin Heidelberg (2010)
14. Devegili, A.J., hÉigeartaigh, C.O., Scott, M., Dahab, R.: Multiplication and Squaring on Pairing-Friendly Fields. Cryptology ePrint Archive, Report 2006/471 (2006)
15. 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 Berlin Heidelberg (2007)
16. Fan, J., Vercauteren, F., Verbauwhede, I.: Faster $\mathbb{F}_p$-Arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves. In: Clavier, C., Gaj, K. (eds.) CHES 2009, LNCS, vol. 5747, pp. 240–253. Springer Berlin Heidelberg (2009)
17. Fuentes-Castañeda, L., Knapp, E., Rodríguez-Henríquez, F.: Faster hashing to G2. In: SAC 2011. pp. 412–430. SAC'11, Springer-Verlag, Berlin, Heidelberg (2012)
18. Gouvêa, C., Oliveira, L., López, J.: Efficient Software Implementation of Public-Key Cryptography on Sensor Networks Using the MSP430X Microcontroller. Journal of Cryptographic Engineering 2(1), 19–29 (2012)
19. Granger, R., Scott, M.: Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions. In: Nguyen, P., Pointcheval, D. (eds.) PKC 2010, LNCS, vol. 6056, pp. 209–223. Springer Berlin Heidelberg (2010)
20. Grewal, G., Azarderakhsh, R., Longa, P., Hu, S., Jao, D.: Efficient Implementation of Bilinear Pairings on ARM Processors. In: SAC 2012. LNCS, Springer Berlin Heidelberg (2013)
21. Hutter, M., Joye, M., Sierra, Y.: Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 170–187. Springer (2011)
22. Hwang, J.Y., Lee, S., Chung, B.H., Cho, H.S., Nyang, D.: Short Group Signatures with Controllable Linkability. In: LIGHTSEC 2011. pp. 44–52. IEEE Computer Society, Washington, DC, USA (2011)
23. IEEE: P1363.3TM/D1 Draft Standard for Identity-based Public-key Cryptography Using Pairings. (2008)
24. Kammler, D., Zhang, D., Schwabe, P., Scharwaechter, H., Langenberg, M., Auras, D., Ascheid, G., Mathar, R.: Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. In: CHES 2009. pp. 254–271. Springer-Verlag, Berlin, Heidelberg (2009)
25. Kiltz, E., Pietrzak, K.: Leakage Resilient ElGamal Encryption. In: Abe, M. (ed.) ASIACRYPT 2010, LNCS, vol. 6477, pp. 595–612. Springer Berlin Heidelberg (2010)
26. Miller, V.S.: The Weil Pairing, and Its Efficient Calculation. Journal of Cryptology 17(4), 235–261 (2004)
27. Miyaji, A., Nakabayashi, M., Takano, S.: New Explicit Conditions of Elliptic Curve Traces for FR-Reduction (2001)

28. Montgomery, P.L.: Modular Multiplication without Trial Division. Mathematics of Computation 44, 519–521 (1985)
29. National Institute of Standards and Technology (NIST): FIPS-186-3: Digital Signature Standard (DSS) (2009), `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`
30. Nogami, Y., Akane, M., Sakemi, Y., Kato, H., Morikawa, Y.: Integer Variable χ-Based Ate Pairing. In: Galbraith, S., Paterson, K. (eds.) Pairing 2008, LNCS, vol. 5209, pp. 178–191. Springer Berlin Heidelberg (2008)
31. Sánchez, A.H., Rodríguez-Henríquez, F.: NEON Implementation of an Attribute-Based Encryption Scheme. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013, LNCS, vol. 7954, pp. 322–338. Springer Berlin Heidelberg (2013)
32. Szczechowiak, P., Kargl, A., Scott, M., Collier, M.: On the Application of Pairing Based Cryptography to Wireless Sensor Networks. In: Basin, D.A., Capkun, S., Lee, W. (eds.) WISEC 2009. pp. 1–12. ACM (2009)
33. Szczechowiak, P., Oliveira, L.B., 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 (2008)
34. Vercauteren, F.: Optimal Pairings. Information Theory, IEEE Transactions on 56(1), 455–461 (2010)
35. Wenger, E.: Hardware Architectures for MSP430-Based Wireless Sensor Nodes Performing Elliptic Curve Cryptography. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013, LNCS, vol. 7954, pp. 290–306. Springer Berlin Heidelberg (2013)
36. Wenger, E., Unterluggauer, T., Werner, M.: 8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors. In: Paul, G., Vaudenay, S. (eds.) INDOCRYPT 2013. LNCS, vol. 8250, pp. 244–261. Springer International Publishing (2013)
37. Wenger, E., Werner, M.: Evaluating 16-Bit Processors for Elliptic Curve Cryptography. In: Prouff, E. (ed.) CARDIS 2011, LNCS, vol. 7079, pp. 166–181. Springer Berlin Heidelberg (2011)

A Optimizations

A.1 Final Exponentiation

The hard part of the final exponentiation by Fuentes-Castañeda *et al.* [17] yields fast execution by reducing the number of multiplications and exponentiations in $\mathbb{F}_{p^{12}}$. As a drawback, it requires four large temporary variables in $\mathbb{F}_{p^{12}}$. In order to attain a low-memory implementation, we decreased the number of temporary variables by adapting their formulas without noticeably degrading performance. Therefore, we initially set $t_0 = f^p$ and compute the chain

$$f^u \rightarrow f^{2u} \rightarrow f^{4u} \rightarrow f^{6u} \rightarrow f^{6u^2} \rightarrow f^{12u^2} \rightarrow f^{12u^3} .$$

Following, a and b are set to $a = f^{6u} \cdot f^{6u^2} \cdot f^{12u^3}$ and $b = a \cdot (f^{2u} \cdot f)^{-1}$. The computation of the result, namely

$$\begin{aligned} f &= f^{6u^2} \cdot f \cdot f^p , \\ f &= [f \cdot a][b]^p[a]^{p^2}[b]^{p^3} , \end{aligned}$$

Algorithm 1 Memory-optimized hard part of the final exponentiation for pairings over BN curves.

```
Input: $f \in \mathbb{F}_{p^{12}}$
Output: $f^{\phi_{12}(p)/n} \in \mathbb{F}_{p^{12}}$
 1: $t_0 \leftarrow f^p$
 2: $b \leftarrow f^u$
 3: if $u < 0$ then $b \leftarrow \overline{b}$    ▷ Conjugate
 4: $b \leftarrow b^2$
 5: $a \leftarrow b^2$
 6: $a \leftarrow a \cdot b$
 7: $b \leftarrow b \cdot f$
 8: $b \leftarrow \overline{b}$
 9: $f \leftarrow f \cdot t_0$
10: $t_0 \leftarrow a^u$
11: if $u < 0$ then $t_0 \leftarrow \overline{t_0}$
12: $f \leftarrow f \cdot t_0$
13: $a \leftarrow a \cdot t_0$
14: $t_0 \leftarrow t_0^2$
15: if $u < 0$ then $t_0 \leftarrow \overline{t_0}$
16: $a \leftarrow a \cdot t_0^u$    ▷ Interleaved
17: $b \leftarrow b \cdot a$
18: $t_0 \leftarrow b^p$
19: $t_0 \leftarrow t_0 \cdot a$
20: $t_0 \leftarrow t_0^p$
21: $t_0 \leftarrow t_0 \cdot b$
22: $t_0 \leftarrow t_0^p$
23: $t_0 \leftarrow t_0 \cdot f$
24: $f \leftarrow t_0 \cdot a$
25: return $f$
```

requires one more multiplication and one more Frobenius action than originally. However, the respective implementation in Algorithm 1 requires three temporary variables instead of four when the exponentiation and the multiplication on Line 16 are done simultaneously using a dedicated function. Since variables in $\mathbb{F}_{p^{12}}$ are large and RAM is more expensive than ROM, this approach aids to keep chip area low.

A.2 Prime-Field Inversion

The parameterized prime $p(u)$ facilitates an optimized exponentiation-based prime-field inversion for positive u that have low Hamming weight. In such cases, the inverse $a^{-1} \in \mathbb{F}_p$ can be expressed as

$$\begin{aligned} a^{-1} \bmod p = a^{p-2} \bmod p &= a^{36u^4+36u^3+24u^2+6u-1} \bmod p \\ &= a^{6u(4u+6u^2(1+u))} \cdot a^{6u-1} \bmod p\,. \end{aligned}$$

Precomputation of the constant $6u - 1$ and the chain of computations

$$a^{6u-1} \rightarrow a^{6u} \rightarrow a^{12u^2} \rightarrow a^{24u^2} \rightarrow a^{36u^2} \rightarrow a^{36u^3} \rightarrow a^{36u^4}$$

enables the computation of the inverse as

$$a^{-1} \bmod p = a^{6u-1} \cdot a^{24u^2} \cdot a^{36u^3} \cdot a^{36u^4} \bmod p\,.$$

Consequently, prime field inversion is done using three fast exponentiations by u, one exponentiation by $6u-1$, five multiplications and two squarings. Since the exponents are fixed and publicly known, Montgomery ladders are not required and runtime thus remarkably benefits from the low Hamming weight of u.