Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves [⋆]

David Kammler[1], Diandian Zhang[1], Peter Schwabe[2],
Hanno Scharwaechter[1], Markus Langenberg[3], Dominik Auras[1],
Gerd Ascheid[1], Rainer Leupers[1], Rudolf Mathar[3], and Heinrich Meyr[1]

[1] Institute for Integrated Signal Processing Systems (ISS),
RWTH Aachen University, Aachen, Germany
`kammler@iss.rwth-aachen.de`

[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, Eindhoven, Netherlands
`peter@cryptojedi.org`

[3] Institute for Theoretical Information Technology (TI),
RWTH Aachen University, Aachen, Germany
`mathar@ti.rwth-aachen.de`

Abstract. This paper presents a design-space exploration of an application-specific instruction-set processor (ASIP) for the computation of various cryptographic pairings over Barreto-Naehrig curves (BN curves). Cryptographic pairings are based on elliptic curves over finite fields—in the case of BN curves a field $\mathbb{F}_p$ of large prime order p. Efficient arithmetic in these fields is crucial for fast computation of pairings. Moreover, computation of cryptographic pairings is much more complex than elliptic-curve cryptography (ECC) in general. Therefore, we facilitate programming of the proposed ASIP by providing a C compiler.
In order to speed up $\mathbb{F}_p$ arithmetic, a RISC core is extended with additional scalable functional units. Because the resulting speedup can be limited by the memory throughput, utilization of multiple data-memory banks is proposed.
The presented design needs 15.8 ms for the computation of the Optimal-Ate pairing over a 256-bit BN curve at 338 MHz implemented with a 130 nm standard cell library. The processor core consumes 97 kGates making it suitable for the use in embedded systems.

Keywords: Application-specific instruction-set processor (ASIP), design-space exploration, pairing-based cryptography, Barreto-Naehrig curves, elliptic-curve cryptography (ECC), $\mathbb{F}_p$ arithmetic.

1 Introduction

Pairings were first introduced to cryptography as a means to break cryptographic protocols based on the elliptic-curve discrete-logarithm problem (ECDLP) [36], [22]. Joux showed in 2000 that they can also be used constructively for tripartite key agreement [30]; Subsequently, different cryptographic protocols have been presented involving

[⋆] This work has been supported by the UMIC Research Centre, RWTH Aachen University. The third author was supported by the European Commission through the ICT Programme under Contract ICT–2007–216499 CACE. Permanent ID of this document: 7e38974d56cc76a7f572f328ee4a3761. Date: July 14, 2009 cryptographic pairings, including identity-based encryption [16] and short digital signatures [17]. A discussion of various applications that would be impossible or very hard to realize without pairings is given in [15].

Cryptographic pairings are based on elliptic curves. To meet both, security requirements and computational feasibility, only elliptic curves with special properties can be considered as basis for cryptographic pairings. State-of-the-art curves for high-security applications are 256-bit Barreto-Naehrig curves (BN curves), introduced in [7]. They achieve 128-bit security according to [4] or 124-bit security according to [21]. Fast arithmetic on these curves demands for fast finite field arithmetic in a field $\mathbb{F}_p$ of prime order p, where p is determined by the curve construction.

Several high-performance software implementations of pairings over BN curves exist for general-purpose desktop and server CPUs [19], [25], [39]. However, the so far only implementation targeting an embedded system was published by Devegili et al. in [19] (updated in [20]) for a Philips HiPerSmart$^{\text{TM}}$ smart card; a complete pairing computation requires 5.17 s at 20.57 MHz, certainly too much time for interactive processes.

This result shows that in order to make state-of-the-art pairing applications available to the embedded domain we need dedicated hardware to accelerate pairing computations. However, the variety and complexity of pairing applications demand for a flexible and programmable solution, that cannot be satisfied by a static hardware implementation. Application-specific instruction-set processors (ASIPs) are a promising candidate to find a good trade-off between these contradicting demands of speed, flexibility and ease of programmability.

This paper shows a design-space exploration of an ASIP for pairing computations over BN curves. We describe how to trade off execution time against area making the ASIP suitable for use in the embedded domain. Dedicated scalable functional units are introduced that speed up general $\mathbb{F}_p$-arithmetic. Moreover, their critical path delay can be modified in order to be integrated with any existing RISC-like architecture without compromising its clock frequency. We show that the speedup from the special functional units is limited by a memory system with a single memory port. Hence, we introduce a memory system utilizing multiple memory banks. The number of banks can be altered without modification to the pipeline or the target architecture tools including the C compiler. This enables fast design-space exploration. The proposed ASIP thus offers a flexible and scalable implementation for pairing applications.

We are—up to our knowledge—the first to implement and time a complete implementation of high-security cryptographic pairings on dedicated specialized hardware.

We would like to thank Jia Huang for supporting the implementation. We furthermore thank Daniel J. Bernstein, Tanja Lange, Ernst Martin Witte and Filippo Borlenghi for suggesting many improvements to our explanations.

Related work. Several architectures for the computation of cryptographic pairings have been proposed in the literature [11–14, 24, 29, 31–34, 42, 43, 45]. All these implemenations use supersingular curves over fields of characteristic 2 or 3. This choice, together with the choice of the underlying fields, achieves only very low security levels, sometimes even below 80 bit. A comparative overview over these architectures is given in Section 4.

Barenghi et al. recently proposed a hardware architecture for cryptographic pairings using curves defined over fields of large prime characteristic [3]. They use a supersingular curve (with embedding degree 2) defined over a 512-bit field and thus achieve only 72-bit security, according to [21].

Another architecture targeting speedup of pairings and supporting fields of large prime characteristic has been proposed in [47]. The instruction set of a SPARC V8 processor is extended for acceleration of arithmetic in $\mathbb{F}_{2^n}$, $\mathbb{F}_{3^m}$ and $\mathbb{F}_p$. However, the focus is put on minor modifications of the datapath resulting in a performance gain for multiplications in $\mathbb{F}_p$ which is two-fold only. Our work focusses rather on significant datapath extensions in order to achieve high speedup for pairings in the embedded domain.

The architectures closest to the one proposed in this paper are dedicated hardware solutions accelerating arithmetic in general $\mathbb{F}_p$ for elliptic-curve cryptography (ECC) [18, 44]. However, these designs have not been reported to be used for complex applications like pairings. A comparison with these architectures is given in Section 4.

Some other architectures for ECC over prime fields only implement modular arithmetic with moduli of a special form that support a particularly fast modular reduction (see i.e. [26]). These approaches are not adequate for pairing-based cryptography where additional properties of the elliptic curves are required. Thus, a detailed comparison with these architectures is omitted here.

Organization of the paper. Section 2 of the paper gives a short overview of cryptographic pairings and Barreto-Naehrig curves. Section 3 describes our approach of an ASIP suitable for pairing computation. In Section 4 we discuss the results. The paper is concluded and future work is outlined in Section 5.

2 Background on cryptographic pairings

We only give a short overview of the notion of cryptographic pairings, a comprehensive introduction is given in [23, chapter IX].

For three groups G_1, G_2 (written additively) and G_3 (written multiplicatively) of prime order r, a cryptographic pairing is a map $e : G_1 \times G_2 \to G_3$, with the following properties:

- Bilinearity:
  $e(kP, Q) = e(P, kQ) = e(P, Q)^k$ for $k \in \mathbb{Z}$.
- Non-degeneracy:
  For all nonzero $P \in G_1$ there exists $Q \in G_2$ such that $e(P, Q) \neq 1$ and for all nonzero $Q \in G_2$ there exists $P \in G_1$ such that $e(P, Q) \neq 1$.
- Computability:
  There exists an efficient algorithm to compute $e(P, Q)$ given P and Q.

We consider the following construction of cryptographic pairings: Let E be an elliptic curve defined over a finite field $\mathbb{F}_p$ of prime order. Let r be a prime dividing the group order $\#E(\mathbb{F}_p) = n$ and let k be the smallest integer, such that $r \mid p^k - 1$. We call k the embedding degree of E with respect to r. Let t denote the trace of Frobenius fulfilling the equation $n = p + 1 - t$.

Let $P_0 \in E(\mathbb{F}_p)$ and $Q_0 \in E(\mathbb{F}_{p^k})$ be points of order r such that $Q_0 \notin \langle P_0 \rangle$, let $\mathcal{O} \in E(\mathbb{F}_p)$ denote the point at infinity. Define $G_1 = \langle P_0 \rangle$ and $G_2 = \langle Q_0 \rangle$. Let $G_3 = \mu_r$ be the group of r-th roots of unity in $\mathbb{F}_{p^k}^*$.

For $i \in \mathbb{Z}$ and $P \in E$ a Miller function [37] is an element $f_{i,P}$ of the function field of E, such that the principal divisor of $f_{i,P}$ is $\mathrm{div}(f_{i,P}) = i(P) - ([i]P) - (i-1)\mathcal{O}$.

Using such Miller functions, we can define the map

$$e_s : G_1 \times G_2 \to \mu_r; (P, Q) \mapsto f_{s,P}(Q)^{(p^k-1)/r}.$$

For certain choices of s the map e_s is non-degenerate and bilinear. For $s = r$ we obtain the reduced-Tate pairing τ and for $s = T = t - 1$ we obtain the reduced-Ate pairing α by switching the arguments [28]. Building on work presented in [35], Vercauteren introduced the Optimal-Ate pairing in [48] which for BN curves can be computed using $s \approx \sqrt{t}$ and a few additional computations (see also [27]).

Using twists of elliptic curves we can further define the generalized reduced-η pairing [28], [5]. In [39] a method to compute the Tate and η pairing keeping intermediate results in compressed form is introduced. We refer to the resulting algorithms as Compressed-Tate and Compressed-η pairing, respectively.

2.1 Choice of an Elliptic Curve

For cryptographic protocols to be secure on the one hand and the pairing computation to be computationally feasible on the other hand, the elliptic curve E must have certain properties: Security of cryptographic protocols based on pairings relies on the hardness of the discrete logarithm problem in G_1, G_2 and G_3. For the 128-bit security level, the National Institute of Standards and Technology (NIST) recommends a prime group order of 256 bit for $E(\mathbb{F}_p)$ and of 3072 bit for the finite field $\mathbb{F}_{p^k}$ [4].

Barreto-Naehrig curves, introduced in [7], are elliptic curves over fields of prime order p with embedding degree $k = 12$. The group order $n = r$ of $E(\mathbb{F}_p)$ is prime by construction, the values p and n can be given as polynomial expressions in an integer u as follows:

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

For our implementation we follow [19] and set u = `0x6000000000001F2D`, yielding two primes $p(u)$ and $n(u)$ of $l = 256$ bit. The field size of $\mathbb{F}_{p^k}$ then has $256 \cdot k = 3072$ bit. Note, that according to [21], a finite fields of size 3072 bit offers only 124-bit security. In this paper we follow the more conservative estimations of [21] and claim only 124-bit security for pairings over 256-bit BN curves.

2.2 Computation of Pairings

The computation of cryptographic pairings consists of two main steps: the computation of $f_{s,P}(Q)$ for Tate and η pairings or of $f_{s,Q}(P)$ when considering the Ate pairing and the final exponentiation with $(p^k - 1)/r$.

The first part is usually done iteratively using variants of Miller's algorithm [37]. Several optimizations of this algorithm have been presented in [6]. The resulting algorithm is often referred to as BKLS algorithm. For BN curves even more optimizations can be applied by exploiting the fact that such curves have sextic twists. A detailed description of efficient computation of pairings over BN curves, including the computation of Miller functions and the final exponentiation is given in [19]. Our implementation follows this description in large parts.

Finite field computations constitute the bulk of the pairing computation – in software implementations typically more than 90% of the time is spent on modular multiplication, inversion and addition. The number of these operations for the implemented pairing algorithms is shown in Table 1.

	number of		
pairing algorithm	*multiplications*	*additions*	*inversions*
Optimal Ate	17,913	84,956	3
Ate	25,870	121,168	2
η	32,155	142,772	2
Tate	39,764	174,974	2
Compressed η	75,568	155,234	0
Compressed Tate	94,693	193,496	0

Table 1. Number of $\mathbb{F}_p$ operations for different pairing applications

Throughout the pairing computation we keep points on elliptic curves in Jacobian coordinates and can thus almost entirely avoid field inversions; our targets for hardware acceleration are thus multiplication and addition in $\mathbb{F}_p$, inversion is implemented as exponentiation with $p - 2$.

3 An ASIP for Cryptographic Pairings

To implement various pairing algorithms (Optimal Ate, Ate, η, Tate, Compressed η and Compressed Tate), a programmable and therefore flexible architecture is targeted in this paper. Standard architectures like embedded RISC cores are flexible, but they are lacking sufficient computational performance for specific applications. Therefore, we apply the ASIP concept to cryptographic-pairing applications in order to reduce the computation time while maintaining programmability. Development and implementation of our ASIP have been carried out using the Processor Designer from CoWare [1]. We used this tool suite for designing the actual architecture implementation on register transfer level (RTL) as well as the simulator and architecture specific software tools.

Keeping control over the data flow on the higher layers of the pairing computation, like $\mathbb{F}_{p^{12}}$ or $E(\mathbb{F}_{p^2})$ arithmetic, is a rather complex task. This calls for a convenient programming model. However, on the lower level realizing the $\mathbb{F}_p$ arithmetic, computational performance is of highest priority. Therefore, we decided to extend a basic 5-stage 32-bit RISC core with special $\mathbb{F}_p$ instructions. Inversions are not considered for special instructions as they are used very seldom ($\leq 3\times$) in any of the targeted applications. The available C compiler enables convenient application development on higher levels, while the computational intensive tasks are mapped to the specialized instructions accessible via intrinsics[4].

Among the targeted $\mathbb{F}_p$ operations, the most challenging to implement is fast modular multiplication, especially for a large word width (e.g. 256 bit). In general, multiplication in $\mathbb{F}_p$ can be done by first multiplying the two factors (256 bit each) and then reducing the product (of 512 bit) modulo p. This might indeed be the fastest approach, if p could be chosen of a special form as for example specified in [40] or [8]. However, due to the construction of Barreto-Naehrig curves (see [7]) we cannot use such primes. Therefore, our approach uses Montgomery arithmetic [38].

[4] An adoption of our code to general purpose processors using the GMP library instead of intrinsics is available from `http://cryptojedi.org/crypto/`.

3.1 Data Processing: A Scalable Montgomery-Multiplier Unit

In 1985 Montgomery introduced an algorithm for modular multiplication of two integers A and B modulo an integer M [38]. The idea of the algorithm is to represent A as $\hat{A} = AR \mod M$ and B as $\hat{B} = BR \mod M$ for a fixed integer $R > M$ with $\gcd(R, M) = 1$. This representation is called Montgomery representation. To multiply two numbers in Montgomery representation we have to compute $\widehat{AB} = \hat{A}\hat{B}R^{-1} \mod M$. For certain choices of R this computation can be carried out much more efficiently than usual modular multiplication: Let us assume that M is odd and let l be the bitlength of M. Choosing $R = 2^l$ clearly fulfills the requirements on R. Division operations during modular reduction can then be replaced by more efficient shifts.

In the context of $\mathbb{F}_p$-multiplication the modulus M corresponds to p. All $\mathbb{F}_p$ operations can be performed in Montgomery representation. Therefore, all values can be kept in Montgomery representation throughout the whole pairing computation.

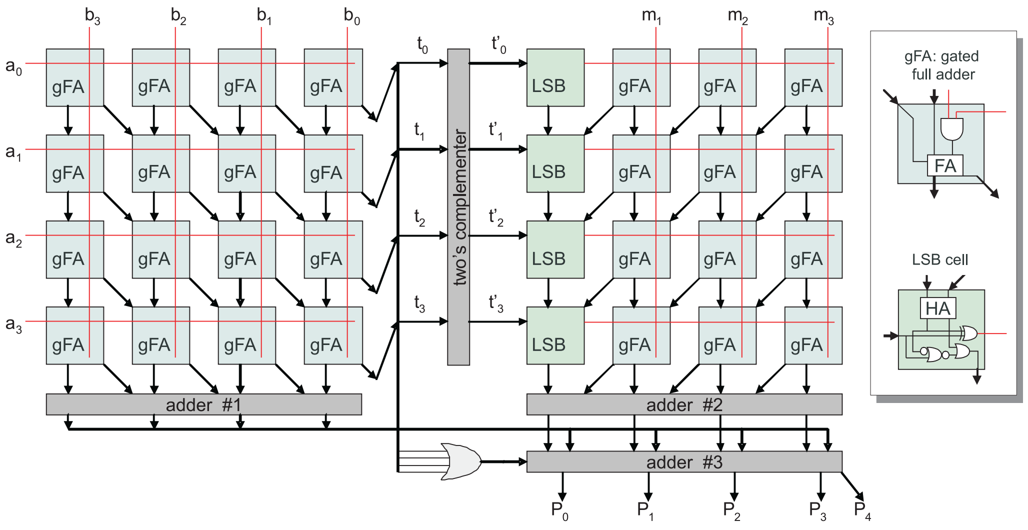

Fig. 1. Montgomery-multiplier based on Nibouche et al. [41]

Nibouche et al. introduced a modified version of the Montgomery multiplication algorithm in [41]. It splits the algorithm into two multiplication operations, that can be carried out simultaneously and use carry save (CS) number representation. This results in a fast architecture that can be pipelined and segmented easily. Therefore, it is chosen as basis for our development. A 4×4-bit example is shown in Fig. 1.

The actual multiplication is carried out in the left half of the architecture, while the reduction is performed in the right part simultaneously. The left part is a conventional multiplier built of gated full adders (gFAs), whereas the right part consists of a multiplier with special cells for the least-significant bits (LSBs). The LSB cells are built around a half adder (HA). Their overall delay is comparable to that of a gFA. A more detailed description can be found in [41].

Due to area constraints we decided to implement only subsets of the regular structures of the multiplier and perform the computation in multiple cycles. The CS-based design provides the opportunity to not only make horizontal but also vertical cuts while the critical path of the multiplier unit depends on its height (H) only. This makes the design *adaptable* to existing cores in terms of *timing* maintaining the performance of their general instruction set. Once the height of the multiplier unit is chosen (in our case $H = 8$), the width (W) can be selected to *adapt* the design to the desired *computational performance* and to trade off *area vs. execution time* of the multiplication. The number and width of required registers to store intermediate data is independent from this choice.

cycle	*partial multiplication ($W \times H$-bit)*	*partial reduction ($W \times H$-bit)*
$t_0 + 0$	$A[\ 7 : \ 0] \times B[255 : 224]$	
$t_0 + 1$	$A[\ 7 : \ 0] \times B[223 : 192]$	
$\vdots$	$\vdots$	
$t_0 + 7$	$A[\ 7 : \ 0] \times B[\ 31 : \ 0]$	
$t_0 + 8$	$A[\ 15 : \ 8] \times B[255 : 224]$	$T'[\ 7 : \ 0] \times M[255 : 224]$
$t_0 + 9$	$A[\ 15 : \ 8] \times B[223 : 192]$	$T'[\ 7 : \ 0] \times M[223 : 192]$
$\vdots$	$\vdots$	$\vdots$
$t_0 + 15$	$A[\ 15 : \ 8] \times B[\ 31 : \ 0]$	$T'[\ 7 : \ 0] \times M[\ 31 : \ 0]$
$\vdots$	$\vdots$	$\vdots$
$t_0 + 248$	$A[255 : 248] \times B[255 : 224]$	$T'[247 : 240] \times M[255 : 224]$
$t_0 + 249$	$A[255 : 248] \times B[223 : 192]$	$T'[247 : 240] \times M[223 : 192]$
$\vdots$	$\vdots$	$\vdots$
$t_0 + 255$	$A[255 : 248] \times B[\ 31 : \ 0]$	$T'[247 : 240] \times M[\ 31 : \ 0]$
$t_0 + 256$	addition #1	$T'[255 : 248] \times M[255 : 224]$
$t_0 + 257$		$T'[255 : 248] \times M[223 : 192]$
$\vdots$		$\vdots$
$t_0 + 263$		$T'[255 : 248] \times M[\ 31 : \ 0]$
$t_0 + 264$		addition #2
$t_0 + 265$	addition #3	

Table 2. Time-flow example for 256-bit multi-cycle modular multiplication ($W = 32, H = 8$)

Multiplication and reduction are carried out simultaneously starting from the most-significant bit (MSB) of their second operand (B and M) first. However, the reduction cannot be started until the incoming data for the LSB cells are available from the two's complementer. Therefore, reduction starts after the first H lines of multiplication have been executed and remains delayed for $\lceil \frac{l}{W} \rceil$ cycles (required for the computation of H lines). Table 2 shows the resulting time flow for a 256-bit modular multiplication with a 32×8-bit unit. Eventually, the CS results need to be transformed back to two's complement number representation (by *addition #1* and *addition #2*) before they are combined to the result by *addition #3*. This is necessary since the result lies in the range of 0 to $2M - 1$, and requires a final comparison against M, which is difficult to handle in CS representation. The comparison including a necessary subtraction of M is performed in another functional unit introduced later. Equation (1) gives the number of required cycles c_{MM} to perform a Montgomery multiplication with the proposed multi-cycle architecture for the general case.

$$c_{MM} = \left(\left\lceil \frac{l}{H} \right\rceil + 1 \right) \cdot \left\lceil \frac{l}{W} \right\rceil + 2 \tag{1}$$

For evaluation, we implemented this multi-cycle Montgomery-multiplier (MMM) in three different sizes ($W \times H$): 32×8 bit, 64×8 bit and 128×8 bit, resulting in an execution time of 266, 134 and 68 cycles respectively. However, the area savings for smaller (and slower) architectures do not scale as well as the execution time. This results from the increased complexity of the required multiplexing for smaller MMM units. In order to keep the amount of multiplexers small, we designed special 256-bit shift registers, that enable a circular shift by W bits for the operands B, M and the corresponding intermediate CS values. This solution is suitable, since the input values are accessed in consecutive order by blocks of W bits. Still, area savings when scaling a 128×8-bit architecture down to 32×8-bit are about 50%.

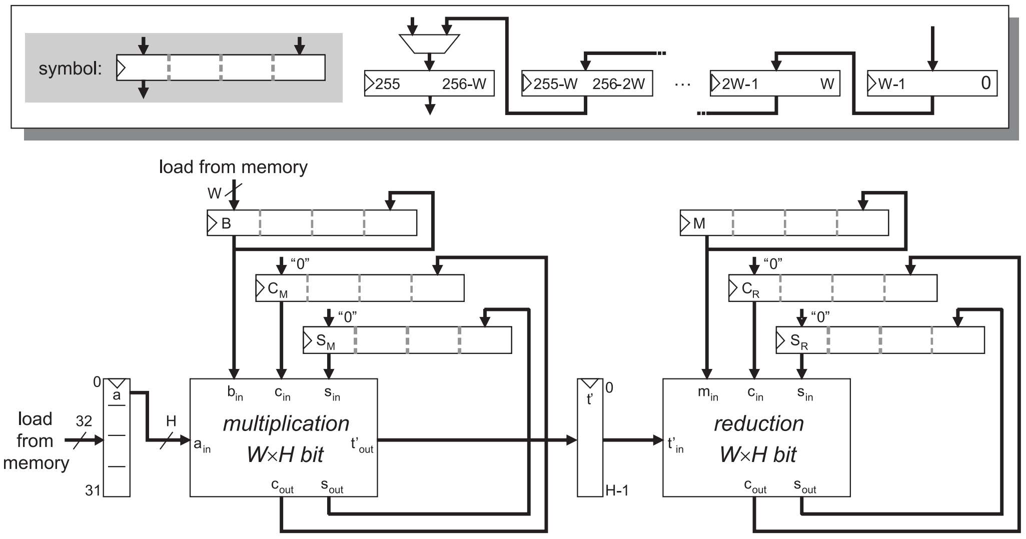

Fig. 2. Structure of the multi-cycle Montgomery-multiplier (MMM)

Fig. 2 shows the overall resulting structure of the MMM unit. The two's complementer is included in the *multiplication unit*, while the *reduction unit* contains additional LSB cells that produce input for the gFA cells on the fly (as depicted in Fig. 1). The input shift registers are initialized step by step during the first $\lceil \frac{l}{W} \rceil$ cycles. After the whole process, the result is stored in the registers for temporary CS values (C_M, S_M, C_R, S_R). The adders for the final summations are not depicted.

An advantage of stepwise executing the multiplication is that the total multiplication width l can be configured at runtime in steps of W. The overall dependence of the execution time on l is quadratic. Modular multiplication is thus significantly faster for smaller multiplication width. This may be interesting for ECC applications requiring lower security.

Similar to the MMM unit we developed a *multi-cycle adder unit* for modular additions and subtractions, which reads two input operands block-wise and simultaneously. For evaluation, a 32-bit and a 64-bit version of this unit have been implemented. Details are omitted here since the implementation is straightforward. Please note that the adder unit causes higher demand on the throughput of data than the MMM, since an addition can be performed within much shorter time.

Both, MMM and *adder unit* require a final subtraction of M whenever the result exceeds this prime number. A special *writeback unit* takes care of this subtraction right before writing back the data, operating block-wise in multiple cycles as well. This unit has been implemented with a width of 32, 64 and 128 bit.

During the execution of multi-cycle operations for modular addition, subtraction and multiplication the pipeline is stalled. Three special instructions are implemented triggering these operations. Instruction arguments are registers containing the starting address of each of the three 256-bit operands. Since the modulus M is not changed during an application run, a special register is utilized and implicitly accessed by the instructions. This register is initialized with p at the beginning of an application via another dedicated instruction.

3.2 Data Access: An Enhanced Memory Architecture

Due to the large width of the operands, the existing 16x32-bit general purpose register file could only store two operands simultaneously. This results in frequent memory accesses consuming additional cycles and thus decreasing the overall performance of the architecture significantly. Enlarging the register file would be very costly in terms of area consumption. Hence, the instructions triggering the multi-cycle operations for modular addition, subtraction and multiplication are implemented as memory-to-memory instructions. This way, the memory accesses can be almost completely hidden in the actual computation.

The resulting throughput demands on the memory system are quite high. Especially the modular addition/subtraction requires a throughput higher than one 32-bit word per cycle. The following two evident mechanisms to increase memory throughput for ASIP designs are not well suited here: First, using memories with multiple ports is costly. The number of ports is limited to two for SSRAMs and the required area is roughly doubled. Second, designing a dedicated system with several (often specialized) memories targets highest performance, but is a complex task. The data memory space gets segmented irregularly, making it difficult to access and manage for a compiler.

Due to the drawbacks of these two approaches we apply a different technique, which we would like to introduce as *transparent interleaved memory segmentation (TIMS)*. Its basic principle is to extend the number of ports to the memory system in order to increase the throughput by using memory banks. These banks are selected on the basis of address bits and can be accessed in parallel. In case of our ASIP, the LSBs of the address are used for the memory bank selection. This results in an addressing scheme, where the memory is selected by calculating the address modulo the number of memories m_d, which has to be a power of two.

In principle, the distribution of accesses to a banked memory system can be handled in software or hardware. However, memory-access conflicts can occur when simultaneous accesses refer to the same memory. Solving these conflicts in software requires an extension of the C compiler in order to avoid multiple simultaneous accesses to the same memory bank. If these conflicts can be ruled out at compile time, this approach results in very efficient code. However, if the conflicts do occur at runtime, additional code to resolve the conflict needs to be included in the target software at the cost of increased execution time. Especially when pointers are used and function calls have a substantial degree of nesting (which is the case for the targeted pairing applications), detecting the conflicts at compile time is often impossible. It requires a significant extension of the C compiler functionality and comes at the cost of increased code size and execution time.

However, due to fairly simple mechanisms and regularity, the distribution of accesses to the memories and resolution of access conflicts can be handled efficiently at runtime by a dedicated hardware block, the *memory-access unit (MAU)* that distributes the memory accesses from the pipeline to the correct memory (Fig. 3).

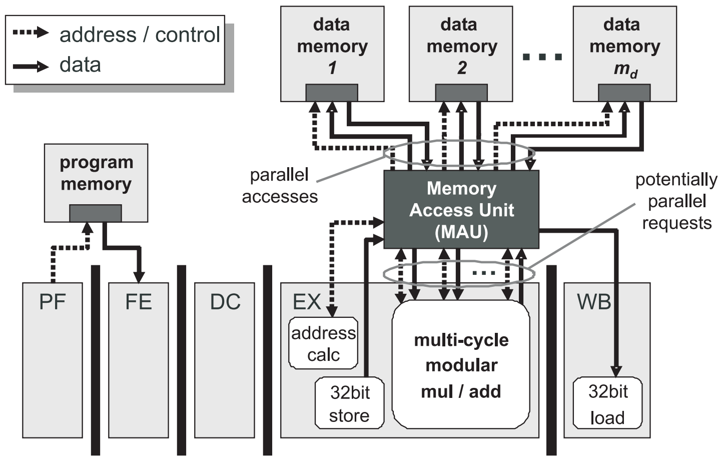

Fig. 3. TIMS implementation with MAU

Memory accesses are requested concurrently by the pipeline on demand resulting in multiple independent read or write connections (unidirectional) between pipeline and MAU. The MAU takes care of granting accesses. Therefore, a simple handshaking protocol is used between pipeline and MAU, which is able to confirm a request within the same cycle in order not to cause any delay cycles when trying to access the fast SSRAMs.

One advantage of this mechanism is the fact, that from the perspective of the core, the memory space remains unchanged, regardless of the number of attached memories. Existing load and store instructions are sufficient to access the whole data memory space. Even when special instructions perform concurrent memory accesses, a modification in the memory system (e.g. changing number of attached memories) does not result in a change of the core or the C compiler. This enables orthogonal implementation and modification of the base architecture and the memory system.

A priority-based hardware resolution of access conflicts is implemented in the MAU in two ways. Static priorities can be used if certain accesses always have higher priority than others. For instance write accesses from later pipeline stages should always have higher priority than read accesses from prior stages. When the priority is changing at runtime, dynamic priority management is required. Then, dedicated additional communication lines between core and MAU indicate a change of priority. In our design this is required for the accesses from the *adder unit*.

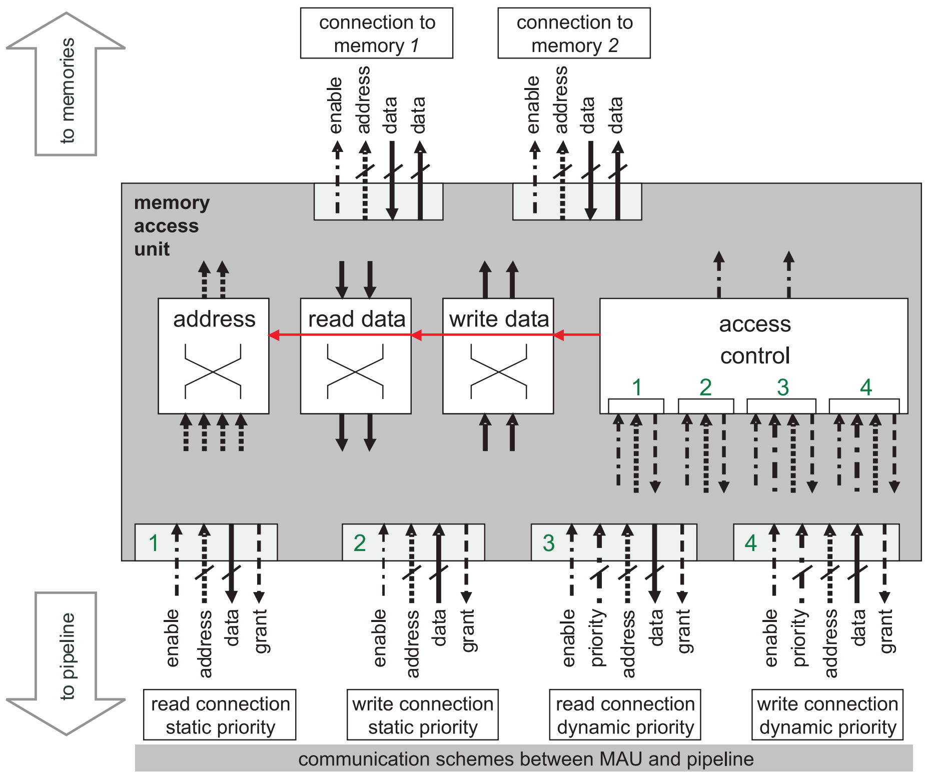

Fig. 4. Interconnect of memory-access unit (MAU)

Fig. 4 depicts the four different connection schemes between MAU and pipeline. The number and type of connections between MAU and pipeline are determined by the number and type of independent memory accesses initiated by the pipeline, while the number of actual memory connections depends on the number of attached data memories m_d ($m_d = 2$ in this example). For sake of clarity, the actual interconnections within the MAU have been omitted in Fig. 4. The *access-control* block combines the *enable* and *priority* signals with the $\log_2(m_d)$ LSBs of the *address* signals from the read and write connections in order to produce the *grants*. At the same time the *enable* signals for the SSRAMs are set accordingly by this unit. It also controls the crossbars that are switching the correct *address*, *read data* and *write data* signals to the memory ports. Please note, that the *read data* crossbar is switched with one cycle delay compared to the *address* crossbar in order to be in sync with the single cycle read latency of the SSRAMs. Effects of TIMS on physical parameters like timing and area consumption are discussed in detail in the result section.

Overall, the TIMS approach enables to extend any existing architecture with memory banks without altering the existing pipeline or the basic instruction set. The specialized memory-to-memory instructions can take full advantage of parallel accesses to these banks reducing execution time. It is not necessary to extend the C compiler of the base architecture with more than intrinsics for the special instructions. The compiled executable can be used on any architecture variant independently from the number of memory banks. As the MAU hides the actual memory system from the pipeline, the number of memory banks can be changed without modification to the pipeline enabling a fast design space exploration.

Still, memory access collisions decrease the performance of the system and cannot be avoided completely due to the automatic address management of the C compiler. However, in our case this effect is kept minimal due to the good distribution of the 256-bit words. For additions and multiplications this causes a maximum additional delay of one cycle only. This results in a maximum performance degradation caused by memory-access conflicts of less than 2% for any of the implemented pairing applications.

	number of data memories (single 32-bit port)		
256-bit $\mathbb{F}_p$-Operation	4 *(128-bit writeback)*	2 *(64-bit writeback)*	1 *(32-bit writeback)*
mod mul (128×8-bit unit)	79	83	91
mod mul (64×8-bit unit)	150 - 151	153	161
mod mul (32×8-bit unit)	294 - 295	296 - 297	301
mod add (64-bit unit)	8 - 9	14 - 15	26
mod add (32-bit unit)	12 - 13	14 - 15	26

Table 3. Number of cycles for 256-bit $\mathbb{F}_p$-arithmetic including memory access

Due to hidden latencies of memory accesses and possible memory-access collisions, the execution times of the arithmetic functional units and the memory system cannot be subsumed in a simple equation. Table 3 presents the cycle count for each operation depending on the number of attached memories. It can be seen that the cycle count for an addition using the 64-bit *adder unit* does not differ from the case utilizing a 32-bit unit, when one or two 32-bit wide memories are attached. The throughput of the memory system limits the maximum achievable performance in this case. Not adapting the width of the adder to the memory system therefore wastes either performance or area, this also holds for the *writeback unit*. Thus the width of the *writeback unit* is coupled to the number of memories in the table. Note that the performance gain of the 32-bit *adder unit* in the four-memories case results from a faster writeback and not from an actual speedup of the addition. The implementation of a 16-bit adder for the single-memory case would not reduce area significantly due to additional multiplexing and is therefore neglected. The width of the multiplier can be selected independently from the number of memories, since operands do not need to be fetched continuously from the memory and smaller memory bandwidth reduces performance only slightly.

4 Results

Overall, we have implemented nine variants of our ASIP with different design parameters regarding number of data memories and width of the computational units for modular multiplication, modular addition and multi-cycle writeback (Table 4). As explained in the previous section, the number of data memories is closely coupled with the width of the *adder* and the *writeback unit*. All synthesis results have been obtained with Synopsys Design Compiler [2] using a 130 nm CMOS standard cell library with a supply voltage of 1.2 V and are given before place and route. The memories are synchronous single-port SRAMs with a latency of one cycle. The total data-memory size is 2048 words for each of the design variants. The program memory is not included in the area reports,

Variant	*128m4*	*64m4*	*32m4*	*128m2*	*64m2*	*32m2*	*128m1*	*64m1*	*32m1*
mod mul size *(bit)*	128×8	64×8	32×8	128×8	64×8	32×8	128×8	64×8	32×8
mod add width *(bit)*	64	64	64	32	32	32	32	32	32
writeback width *(bit)*	128	128	128	64	64	64	32	32	32
# data memories	4	4	4	2	2	2	1	1	1
total area[a] *(kGates)*	195	186	182	164	153	148	145	134	130
core area[b] *(kGates)*	96	87	83	97	86	81	93	83	79
timing *(ns)*	3.69	3.65	3.52	2.96	2.97	3.02	2.95	3.03	3.09
Optimal Ate *(ms)*	17.5	21.8	29.9	15.8	19.4	27.3	19.2	23.4	32.0
Ate *(ms)*	25.3	31.4	42.6	22.8	27.9	38.9	27.6	33.5	45.6
η *(ms)*	32.3	39.5	52.8	28.8	35.0	48.1	34.6	41.6	56.2
Tate *(ms)*	38.5	47.0	62.7	34.4	41.6	57.1	41.1	49.5	65.3
Compressed η *(ms)*	38.6	55.0	86.2	34.5	48.2	77.1	41.6	56.5	85.8
Compressed Tate *(ms)*	48.2	68.9	107.8	43.2	60.3	96.5	52.0	70.7	107.3

[a] Including area for data memories

[b] Without area for memories, but including area for MAU

Table 4. Implemented design variants of the ASIP for pairings

since it is not changing through the different designs and could be implemented differently (as ROM, RAM, synthesized logic etc.) depending on the final target system. The plain RISC (32-bit, 5-stage pipeline, 32-bit integer multiplier) without memories and extensions consumes 26 kGates and achieves a timing of 2.89 ns.

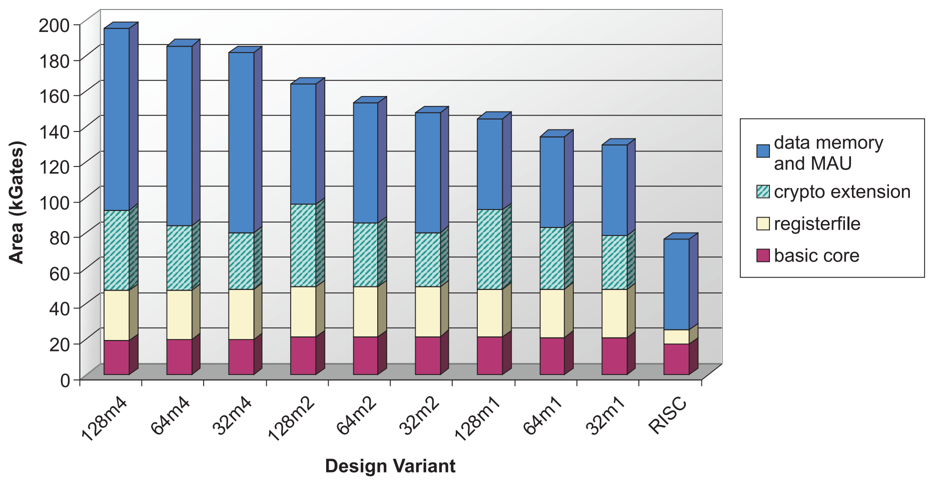

Fig. 5. ASIP area consumption and distribution

Fig. 5 shows the area distribution of the different ASIP variants. While the basic core only shows moderate area increase from 17 to 21 kGates for all variants (resulting from decoder extensions and additional pipeline registers), the area for the register file increases from 9 to 28 kGates compared to the plain RISC. The reason are specialized 256-bit registers storing the prime number and intermediate results of the modular operations. These registers are independent from the width of any of the additional functional units. The area of the cryptographic extensions is dominated by the MMM unit.

Observe that splitting the memory into two of half the size results in a data-memory area increase of 31%. Utilizing a dual port memory instead would increase area by over 83%. The area overhead due to the MAU lies between only 0.5 and 1.2 kGates, when two memories are attached. Even for four attached memories it is below 3.5 kGates.

However, limitations of TIMS utilizing the proposed MAU become visible when looking at the timing of the different variants of the ASIP. While attaching one or two data memories barely affects the critical path with respect to the original RISC architecture (within design tool accuracy, see Table 4), an increased delay is observed when four memories are attached. This delay is caused by the complexity of priority resolution for four attached memories combined with four independent memory accesses with dynamic priority, which are necessary to implement the 64-bit adder.

The execution times of all six implemented pairing applications on all nine ASIP variants are shown in Table 4. Note that running an application on different ASIP variants does not require recompiling the application because the instruction set is identical for all of them. For all applications performance improves significantly with increasing width of the MMM. Also, the number of cycles decreases when increasing the number of connected data memories. Unfortunately, the longer critical path of the four-memory system leads to a lower performance than for the designs with two memories. The overall fastest design is variant *128m2*, executing the Optimal-Ate pairing in 15.8 ms. With the smallest and slowest variant completing the task in 32.0 ms, the user is offered a quite broad design space enabling trade-offs.

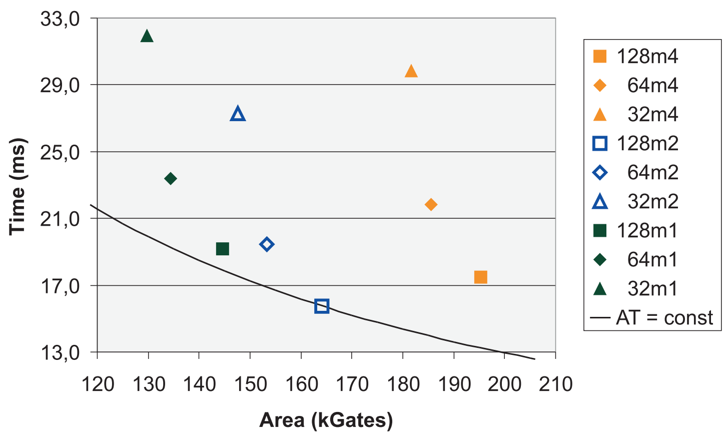

Fig. 6. Area-time trade-off for different ASIP variants (Optimal Ate pairing)

In order to evaluate the efficiency of the different design variants, Fig. 6 shows the area-time trade-off for the Optimal-Ate pairing. It can be seen clearly that the best AT product is obtained by the *128m2* design. This shows the importance of investigaing the memory architecture of ASIPs during design-space exploration. In our case the best results are obtained with TIMS and *two* data memories in spite of the considerable area increase due to the memory splitting.

4.1 Performance Comparison

To our best knowledge there exists no literature reporting performance figures resulting from actual implementations of cryptographic pairings on dedicated hardware achieving a 124-bit security level. Hardware implementations for lower security levels can obviously be much faster than the proposed design. Table 5 gives an overview of performance and area consumption for various pairing implementations on dedicated hardware; the given security levels are according to [21]. Whenever more than one design variant is given in a publication, the fastest one with the highest security level is listed in the table.

Design	*Technology*	*Area*	*Freq. (MHz)*	*Pairing*	*Field Characteristic*	*Security Level (bit)*	*Time (ms)*
this work	130 nm std. cell	97 kGates	338	Opt. Ate	256-bit prime	124	15.8
[11]	Xilinx xc2vp20	8 kSlices	90	Tate	3	97	0.298
[11]	Xilinx xc2vp20	8 kSlices	115	Tate	2	97	0.327
[43]	Xilinx xc2vp100	44 kSlices	33	Tate	2	80	0.146
[45] upd. [46]	Xilinx xc2vp100	38 kSlices	72	Tate	2	76	0.049
[31]	Xilinx xc2v6000	25 kSlices	47	Tate	2	76	2.81
[32]	Xilinx xc2v6000	15 kSlices	40	Tate	2	76	3.0
[3]	Xilinx xc2v8000	34 kSlices	135	Tate	512-bit prime	72	1.61
[29]	Xilinx xc4vlx200	74 kSlices	199	η_T	3	68	0.008
[14]	Altera ep2c35	19 kLEs	147	η_T	3	68	0.027
[13]	180 nm std. cell	194 kGates	200	η_T	3	68	0.047
[12]	Xilinx xc4vlx15	2 kSlices	203	η_T	3	68	0.137
[42]	Xilinx xc2vp100	15 kSlices	85	η_T	3	68	0.183
[34]	Xilinx xc2vp200	14 kSlices	77	Tate	3	68	0.251
[24]	Xilinx xc2vp4	4 kSlices	150	Tate	3	68	0.432
[33]	Xilinx xc2vp125	56 kSlices	15	Tate	3	68	0.85

Table 5. Performance and area comparison for pairings

The results listed in Table 5 are hardly comparable to the design proposed in this paper. Not only do they achieve lower security levels, they also mainly focus on FPGAs rather than standard cells and mostly use curves over binary or ternary fields. In the following we therefore give a comparison with standard cell designs which accelerate $\mathbb{F}_p$ arithmetic for elliptic-curve cryptography and finally discuss our design in the context of smart cards.

Comparison with standard cell designs for ECC. Other publications describing dedicated-hardware implementations for ECC over fields of large prime characteristic give performance figures in terms of time needed for a scalar multiplication with a scalar k of a certain size, e.g. the computation of $[k]P$ for some $P \in E(\mathbb{F}_p)$. An overview is presented in Table 6.

In order to compare the results of this work with these architectures we implemented scalar multiplication on the 256-bit Barreto-Naehrig curve that we also used for pairing computation. Our design does not accelerate field inversion through hardware, so we use Jacobian projective coordinates to represent the points on the curve, trading inversions for several multiplications.

Design	*Technology*	*Area*	*Freq. (MHz)*	*Scalar Mult. Alg.*	$\log_2(\lvert\mathbb{F}_p\rvert)$	*Time (ms)*
this work	130 nm std. cell	97 kGates	338	NAF recoding	256	0.998
[18]	130 nm std. cell	122 kGates	556	NAF recoding	256	1.01
[44]	130 nm std. cell	107 kGates	138	NAF recoding	256	2.68

Table 6. Performance and area comparison for scalar multiplication

We emphasize that the choice of this curve is far from being optimal in terms of achievable performance for ECC not involving pairings. Clearly, using an elliptic curve in Edwards form [9] and representing points in inverted Edwards coordinates [10] would improve speed for scalar multiplication significantly.

A scalar multiplication with a 256-bit scalar takes 0.998 ms for the *128m2* variant of the proposed design. This number includes transformation of the scalar into NAF and a transformation from Jacobian into affine coordinates at the end. Note that ASIP variant *128m2* is not only slightly faster than the designs in [44] and [18], but also consumes less area. However, it should be noted that the utilization of two data memories in our design affects the overall area consumption.

Application to smart cards. In [19] (updated in [20]), Devigili et al. report 5.17 s for the computation of the Ate pairing over a 256-bit Barreto-Naehrig curve on a Philips HiPerSmart$^{\text{TM}}$ smart card operating at 20.57 MHz. This smart card contains a SmartMIPS-based 32-bit architecture and is manufactured in 180 nm technology. For interactive processes this execution time is not sufficient even when the smart card operates at its maximum frequency of 38MHz. Our design achieves—synthesized in a 180 nm CMOS standard cell library with a supply voltage of 1.8 V—over 230 MHz. Even running our smallest design variant *32m1* at the clock speed of 20.57 MHz (leaving a substantial margin for place and route and implementation of protection mechanisms against side channel attacks), the Ate pairing takes 0.71 s and the Optimal Ate pairing is executed in 0.50 s, which is already sufficient for interactive processes. Depending on the design variant used, speedups of over 20$\times$ could be achieved. This gives an impression of the achievable performance increase for the computation of cryptographic pairings in the embedded domain when more specialized hardware is used.

5 Conclusion and Outlook

In this paper we presented a design-space exploration of an ASIP for computation of cryptographic pairings over BN curves. The design is based on extensions of an existing RISC core, which are completely transparent and independent from the original pipeline. Therefore, they could be applied to any RISC-like architecture, which can stall the pipeline during multi-cycle operations. The extensions are adaptable in terms of timing and enable a trade-off between execution time and area. A flexible and transparent memory architecture extension making use of multiple memories (TIMS) enables fast design space exploration and the usage of existing compilers, since the address space remains unsegmented. We are—up to our knowledge—the first to implement and time a complete implementation of high-security cryptographic pairings on dedicated specialized hardware.

Future objectives are including countermeasures against side-channel attacks, which are not specially targeted in the current design, either in hard- or in software.

References

1. CoWare Processor Designer. `http://www.coware.com/products/processordesigner.php`.
2. Synopsys Design Compiler. `http://www.synopsys.com/products/logic/design_compiler.html`.
3. A. Barenghi, G. Bertoni, L. Breveglieri, and G. Pelosi. A FPGA coprocessor for the cryptographic Tate pairing over $\mathbb{F}_p$. In *Proc. Fifth Int'l Conf. Information Technology: New Generations – ITNG 2008*, pages 112–119, 2008.
4. E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management – part 1: General (revised). National Institute of Standards and Technology, NIST Special Publication 800-57, 2007.
5. P. S. L. M. Barreto, S. D. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. *Designs, Codes and Cryptography*, 42(3):239–271, 2007.
6. P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology – CRYPTO 2002*, pages 354–368, 2002.
7. P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography*, volume 3897 of *LNCS*, pages 319–331, 2006.
8. D. J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography – PKC 2006*, volume 3386 of *LNCS*, pages 207–228, 2006.
9. D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 29–50, 2007.
10. D. J. Bernstein and T. Lange. Inverted Edwards coordinates. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 4851 of *LNCS*, pages 20–27, 2007.
11. J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodríguez-Henríquez. A comparison between hardware accelerators for the modified Tate pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. In *Pairing-Based Cryptography – Pairing 2008*, volume 5209 of *LNCS*, pages 297–315, 2008.
12. J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi. Algorithms and arithmetic operators for computing the η_t pairing in characteristic three. *IEEE Trans. Comput.*, 57(11):1454–1468, 2008.
13. J.-L. Beuchat, H. Doi, K. Fujita, A. Inomata, A. Kanaoka, M. Katouno, M. Mambo, E. Okamoto, T. Okamoto, T. Shiga, M. Shirase, R. Soga, T. Takagi, A. Vithanage, and H. Yamamoto. FPGA and ASIC implementations of the η_t pairing in characteristic three. Cryptology ePrint Archive, Report 2008/280, 2008. `http://eprint.iacr.org/2008/280`.
14. J.-L. Beuchat, M. Shirase, T. Takagi, and E. Okamoto. An algorithm for the η_t pairing calculation in characteristic three and its hardware implementation. In *Proc. 18th IEEE Symp. Computer Arithmetic – ARITH 2007*, pages 97–104, 2007.
15. D. Boneh. A brief look at pairings based cryptography. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science – FOCS 2007*, pages 19–26, 2007.
16. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229, 2001.
17. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, 2004.
18. G. Chen, G. Bai, and H. Chen. A high-performance elliptic curve cryptographic processor for general curves over GF(p) based on a systolic arithmetic unit. *IEEE Trans. Circuits and Systems II: Express Briefs*, 54(5):412–416, 2007.
19. A. J. Devegili, M. Scott, and R. Dahab. Implementing cryptographic pairings over Barreto-Naehrig curves. In *Pairing-Based Cryptography – Pairing 2007*, volume 4575 of *LNCS*, pages 197–207, 2007.
20. A. J. Devegili, M. Scott, and R. Dahab. Implementing cryptographic pairings over Barreto-Naehrig curves. Cryptology ePrint Archive, Report 2007/309, 2007. `http://eprint.iacr.org/2007/390`.
21. M. Näslund (editor). Ecrypt yearly report on algorithms and keysizes (2007-2008), 2008. `http://www.ecrypt.eu.org/ecrypt1/documents/D.SPA.28-1.1.pdf`.

22. G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. of Computation*, 62(206):865–874, 1994.
23. S. Galbraith. Pairings. In I. F. Blake, G. Seroussi, and N. P. Smart, editors, *Advances in Elliptic Curve Cryptography*, London Mathematical Society Lecture Note Series, chapter IX. Cambridge University Press, 2005.
24. P. Grabher and D. Page. Hardware acceleration of the Tate pairing in characteristic three. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 398–411, 2005.
25. P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. ePrint, 2008/205, 2008.
26. T. Güneysu and C. Paar. Ultra high performance ECC over NIST primes on commercial FPGAs. In *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *LNCS*, pages 62–78, 2008.
27. F. Hess. Pairing lattices. In *Pairing-Based Cryptography – Pairing 2008*, volume 5209 of *LNCS*, pages 18–38, 2008.
28. F. Hess, N. P. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Trans. Information Theory*, 52(10):4595–4602, 2006.
29. J. Jiang. Bilinear pairing (Eta_T pairing) IP core. Technical report, 2007. `http://www.cs.cityu.edu.hk/~ecc/doc/etat_datasheet_v2.pdf`.
30. A. Joux. A one round protocol for tripartite Diffie-Hellman. In *Algorithmic Number Theory*, volume 1838 of *LNCS*, pages 385–394, 2000.
31. M. Keller, T. Kerins, F. Crowe, and W. Marnane. FPGA implementation of a GF(2^m) Tate pairing architecture. In *Reconfigurable Computing: Architectures and Applications*, volume 3985 of *LNCS*, pages 358–369, 2006.
32. M. Keller, R. Ronan, W. Marnane, and C. Murphy. Hardware architectures for the Tate pairing over GF(2^m). *Computers & Electrical Eng.*, 33(5-6):392–406, 2007.
33. T. Kerins, W. P. Marnane, E. M. Popovici, and P. S. L. M. Barreto. Efficient hardware for the Tate pairing calculation in characteristic three. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 412–426, 2005.
34. G. Kömürcü and E. Savas. An efficient hardware implementation of the Tate pairing in characteristic three. In *Proc. Third Int'l Conf. Systems – ICONS 2008*, pages 23–28, 2008.
35. E. Lee, H.-S. Lee, and C.-M. Park. Efficient and generalized pairing computation on Abelian varieties. Cryptology ePrint Archive, Report 2008/040, 2008. `http://eprint.iacr.org/2008/040`.
36. A. J. Menezes, T. Okamoto, and S. A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Information Theory*, 39(5):1639–1646, 1993.
37. V. S. Miller. The Weil pairing, and its efficient calculation. *J. Cryptology*, 17:235–261, 2004.
38. P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
39. M. Naehrig, P. S. L. M. Barreto, and P. Schwabe. On compressible pairings and their computation. In *Progress in Cryptology – AFRICACRYPT 2008*, volume 5023 of *LNCS*, pages 371–388, 2008.
40. National Institute of Standards and Technology, NIST. *FIPS 186-2: Digital Signature Standard (DSS)*, 2000. `http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf`.
41. O. Nibouche, A. Bouridane, and M. Nibouche. Architectures for Montgomery's multiplication. *IEE Proc. – Computers and Digital Techniques*, 150(6):361–368, 2003.
42. R. Ronan, C. Murphy, T. Kerins, C. Ó hÉigeartaigh, and P. S. L. M. Barreto. A flexible processor for the characteristic 3 η_t pairing. *Int'l J. High Performance Systems Architecture*, 1(2):79–88, 2007.
43. R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, and T. Kerins. FPGA acceleration of the Tate pairing in characteristic 2. In *Proc. IEEE Int'l Conf. Field Programmable Technology*, pages 213–220, 2006.

44. A. Satoh and K. Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Trans. Computers*, 52(4):449–460, 2003.
45. C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystems over binary fields. In *Proc. IEEE Int'l Conf. Field Programmable Technology – FPT 2006*, pages 173–180, 2006.
46. C. Shu, S. Kwon, and K. Gaj. Fpga accelerated tate pairing based cryptosystems over binary fields. Cryptology ePrint Archive, Report 2006/179, 2006. `http://eprint.iacr.org/2006/179`.
47. T. Vejda, D. Page, and J. Großschädl. Instruction set extensions for pairing-based cryptography. In *Pairing-Based Cryptography – Pairing 2007*, volume 4575 of *LNCS*, pages 208–224, 2007.
48. F. Vercauteren. Optimal pairings. Cryptology ePrint Archive, Report 2008/096, 2008. `http://eprint.iacr.org/2008/096`.