Efficient Elliptic-Curve Cryptography using Curve25519 on Reconfigurable Devices

Pascal Sasdrich, Tim Güneysu

Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum
Bochum, Germany
{pascal.sasdrich,tim.gueneysu}@rub.de

Abstract. Elliptic curve cryptography (ECC) has become the predominant asymmetric cryptosystem found in most devices during the last years. Despite significant progress in efficient implementations, computations over standardized elliptic curves still come with enormous complexity, in particular when implemented on small, embedded devices. In this context, Bernstein proposed the highly efficient ECC instance Curve25519 that was shown to achieve new ECC speed records in software providing a high security level comparable to AES with 128-bit key. These very tempting results from the software domain have led to adoption of Curve25519 by several security-related applications, such as the NaCl cryptographic library or in anonymous routing networks (nTor). In this work we demonstrate that even better efficiency of Curve25519 can be realized on reconfigurable hardware, in particular by employing their Digital Signal Processor blocks (DSP). In a first proposal, we present a DSP-based single-core architecture that provides high-performance despite moderate resource requirements. As a second proposal, we show that an extended architecture with dedicated inverter stage can achieve a performance of more than 32,000 point multiplications per second on a (small) Xilinx Zynq 7020 FPGA. This clearly outperforms speed results of any software-based and most hardware-based implementations known so far, making our design suitable for cheap deployment in many future security applications.

Keywords: FPGA, DSP, ECC, Curve25519, Diffie-Hellman, Xilinx, Zynq

1 Introduction

With the advent of ubiquitous computing, many applications require an appropriate security level, including complex cryptography. In addition, to avoid bottlenecks due to security requirements, those devices often need to provide a significant amount of cryptographic operations per second regardless the computational constraints of the devices. In particular, modern public key cryptosystems, e.g., RSA, use complex and computation-intensive calculations that are often too slow on embedded hardware. Neal Koblitz and Victor Miller proposed independently in 1985 [11, 8] the use of Elliptic Curve Cryptography (ECC) providing similar security compared to RSA but using smaller keys. This benefit allows for greater efficiency when using ECC (160–256 bit) compared to RSA or discrete logarithm schemes over finite fields (1024–4096 bit) while providing an equivalent level of security [9]. In the last years, ECC has become the standard for high-performance asymmetric cryptography, although it still places a high demand on small microprocessors and microcontrollers. Due to this, dedicated cryptographic hardware such as Field-Programmable Gate Arrays (FPGA) are still the preferred choice when high-performance on cryptographic operations is a strict requirement for an embedded application.

Contribution: In this work, we present the first efficient implementation of a special Elliptic Curve Cryptosystem using the Curve25519 [2] on reconfigurable hardware to provide a Diffie-Hellman key agreement suitable for use in high-performance application. It provides inherently timing resistance against simple power attacks (SPA) and a security level comparable to NIST P-256 ECC or AES with 128-bit key. In particular, our design takes advantage of the special-purpose DSP slices of reconfigurable devices in order to increase efficiency and performance. Although ECC using Curve25519 for a Diffie-Hellman key agreement was initially proposed to accelerate primarily its implementation in software, we show that similarly its special characteristics can be exploited in hardware to achieve a compact, space-saving high-performance ECC processor on reconfigurable devices. Our multi-core Curve25519 implementation on a small Xilinx Zynq 7020 FPGA achieves more than 32,000 point multiplications per second with which we can virtually support any high-performance application of asymmetric cryptography.

Outline: The paper is organized as follows: Section 2 provides relevant previous work while Section 3 outlines the Curve25519 function for Diffie-Hellman based key agreement and its special characteristics. Section 4 describes design considerations and decisions, in particular with respect to the arithmetic units. Section 5 describes two different architectures for moderate and high-performance that are evaluated and compared in Section 6. Finally, we conclude our work in Section 7.

2 Previous Work

We briefly summarize previously published results of relevance to this contribution. Since there is a wealth of publication addressing ECC hardware architectures, we refer to the overview in [3] and restrict the discussion of previous works to the most relevant ones. As one of the first works in ECC implementations, Orlando and Paar [13] proposed a design targeting explicitly reconfigurable hardware using Montgomery-based multiplications included with a series of pre-computations. This publication was followed by many more, e.g., [7] trying to improve performance on FPGAs by the use of dedicated multipliers or [15] trying to improve the performance in an algorithmic approach. Using integrated Digital Signal Processors (DSP) both for modular multiplication and modular addition was initially proposed in [14] targeting standardized NIST primes P-224 and P-256 using a special reduction scheme.

3 Background

In the following, we will briefly introduce to the mathematical background relevant for this work. We will start with a short review of the Elliptic Curve Cryptosystems (ECC). Please note that only ECC over prime fields $GF(p)$ will be subject of this work.

Let p be a prime with $p > 3$ and $\mathbb{F}_p = GF(p)$ the Galois Field over p. Given the Weierstrass equation of an elliptic curve

$$\mathcal{E} : y^2 = x^3 + ax + b,$$

with $a, b \in GF(p)$ and $4a^3 + 27b^2 \neq 0$, points $\mathcal{P}_i \in \mathcal{E}$, we can compute tuples (x, y) also considered as points on this elliptic curve $\mathcal{E}$. Based on a group of points defined over this curve, ECC arithmetic defines the addition $\mathcal{P}_3 = \mathcal{P}_1 + \mathcal{P}_2$ of two points $\mathcal{P}_1, \mathcal{P}_2$ using the *tangent-and-chord* rule as the primary group operation. This group operation distinguishes the case for $\mathcal{P}_1 = \mathcal{P}_2$ (*point doubling*) and $\mathcal{P}_1 \neq \mathcal{P}_2$ (*point addition*). Furthermore, formulas for these operations vary for affine and projective coordinate representations, i.e., the curve equation for projective coordinates relaxes the one shown above by introducing another variable Z. The use of projective coordinates allows to avoid the costly modular inversion for a point operation at the cost of additional modular multiplications.

Most ECC-based cryptosystems rely on the Elliptic Curve Discrete Logarithm Problem (ECDLP) and thus employ the technique of point multiplication $k \cdot \mathcal{P}$ as cryptographic primitive, i.e., a k times repeated point addition of a base point $\mathcal{P}$. Precisely, the ECDLP is the fundamental cryptographic problem used in protocols and crypto schemes like the Elliptic Curve Diffie-Hellman key exchange [4], the ElGamal encryption scheme [6] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [1]. Note that all these cryptosystems solely employ affine coordinates to guarantee unique solutions, i.e., in case the faster projective coordinates are used for intermediate computations, a final modular inversion needs to be performed to convert projective coordinates back to affine coordinates.

In this work we will focus on the Diffie-Hellman key exchange based on the special elliptic curve Curve25519 which characteristics are given in the following.

3.1 Curve25519 Function

Curve25519 can be considered a function designed to simplify and accelerate the Diffie-Hellmann key agreement over elliptic curves. It defines a point multiplication that provides inherent timing-attack resistance and a conjectured security level comparable to NIST P-256 or AES-128. Using this function, two parties can derive a shared secret with the help of their 32-byte secret key and the public key.

Curve25519 is based on a prime field with a prime close to a power of 2 (Pseudo Mersenne Prime) and defined as follows:

$$E : y^2 = x^3 + 486662x^2 + x \bmod 2^{255} - 19$$

Assume a base point P with $\mathcal{Q} = (X_i, Y_i, Z_i)$. Tracking two intermediate points $\mathcal{Q}, \mathcal{Q}'$ and their difference $\mathcal{Q} - \mathcal{Q}'$ based on P, Curve25519 defines a combined point doubling and point addition function as a single step of the Montgomery Power ladder [12] with 4 squarings, 5 general multiplications, 1 multiplication by $(A - 2)/4$ and 8 additions or subtractions. An additional benefit of the Curve25519 function is, that only the x-coordinate of each point is required during the computation so that y can be completely omitted in the formulas. Therefore, the point multiplication solely relies on the x and z-coordinate of the two points Q and Q'. Eventually, a combined step is computed by

$$\begin{aligned}
x_2 &= (x^2 - z^2)^2 = (x - z)^2(x + z)^2 \\
z_2 &= 4xz(u^2 + Axz + z^2) \\
x_3 &= 4(xx' - zz') \\
z_3 &= 4(xz' - zx')x_1
\end{aligned}$$

3.2 Curve25519 Computations

For the entire scalar multiplication, computing $k \times P$, a total of 255 step function calls (combined point double and addition) are executed followed by a final inversion and a single multiplication calculating $X \times Z^{-1}$.

Figure 1 shows the flow of the algorithm for three points Q, Q' and Q_1 represented in terms of projective coordinates and where Q_1 is $Q - Q'$. Note, that every addition or subtraction is followed by a multiplication and nearly every multiplication is followed by an addition or subtraction. This observation is very helpful for designing an efficient hardware architecture.

In addition, always the same number of operations are performed, independently of the processed data. Therefore, the computation can be done in constant time preventing timing-based attacks.

4 Design Considerations

For most modern standardized elliptic curves, e.g., the NIST P-256, the underlying prime field is based on a Generalized Mersenne Prime which allows a reduction based on few additions and subtractions. For Curve25519 the field is based on a Pseudo Mersenne Prime $2^n - c$ based on a similar but slightly different concept. For the Curve25519 elliptic curve, the reduction can be computed by a multiplication with a small constant, in this case the constant $c = 19$.

However, this prime with a total of 255 bits has some interesting properties since all field elements can be divided into fifteen words of 17 bit width. Processing these chunks is usually inefficient for common processors which operate

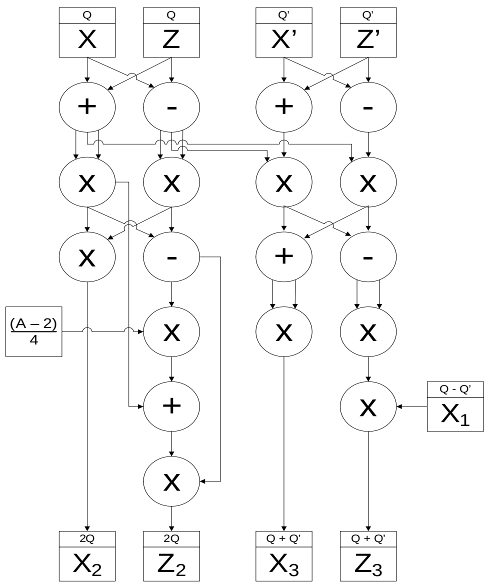

Fig. 1. Double-and-Add formula according to Montgomery's ladder.

on 8-, 16-, 32- or 64-bit data words. Recent FPGA devices, however, provide a multitude of dedicated, full-custom DSP slices equipped with an addition, a multiplication and an accumulation stage for integers enhanced with additional register stages to operate at full device speed. Since the multiplication of DSP blocks natively supports signed 25 × 18-bit wide operands, this is a perfect fit to our requirement of processing unsigned 17-bit data words. By means of an interleaved multiplication schedule, multiplying two 255-bit field elements can be rearranged over several parallelly operating DSP-blocks so that each DSP block has to compute one 17 × 17 bit multiplication at a time resulting in one partial product. The full multiplication can then be performed using 15 DSP slices. Additionally, we can use the included accumulation stage per DSP block to add up the intermediate results. Finally, we end up generating an intermediate result in the accumulation stage that is slightly too large but which can be reduced in a subsequent recombination step. The recombination step itself can be implemented by a constant multiplier with $c = 19$, realignment logic to recombine shares of partial products as well as a subtraction stage to correct the result by reducing it modulo P in case it is slightly too large.

Besides the modular multiplication, point doublings and point additions require the computation of modular additions and subtractions. To return a unique result, a final inversion is required to convert the projective coordinates to affine coordinates. The addition respectively subtraction unit can be implemented as cascade of two DSP blocks, one for the main operation and one for the subsequent reduction. For the basic version of an inversion, we plan to use Fermat's Little Theorem. This approach requires solely a modular multiplier which is already provided by the core despite of a small additional state machine. Although inversion based on this approach hardly requires extra hardware, inversion performance will be comparably slow. Another approach would be to implement a dedicated inversion unit based on the Binary Extended Euclidean Algorithm. This obviously requires a significant amount of additional resources but the computation of the inversion would be significantly faster. Optionally, in a multi-core scenario one dedicated inverter can be shared among several point multiplication cores reducing the overhead costs by resource sharing.

Finally, we plan to use two 36k true dual-port block RAMs (BRAM) to store all intermediate values in a butterfly-wise dataflow. Since 17-bit or 34-bit input and output values need to be processed by arithmetic units, we will specify 34-bit wide ports of the BRAMs.

5 Implementation

In this section, we present first details of a single-core Curve25519 architecture resulting in moderate performance at low resource costs. Second, we extend our design into a multi-core architecture that aims at consuming all available resources of the Xilinx Zynq 7020 device but which is capable to provide maximum throughput.

5.1 Single-Core Architecture

Our single-core Curve25519 implementation is designed to support asymmetric cryptography as a supplementary function and saves most of the FPGA logic for the main application. The cryptographic core is capable to perform a point multiplication in projective coordinates. For use with most cryptographic protocols, the core also needs to implement a final inversion to convert the output in projective coordinates back to affine coordinates. The arithmetic processor therefore supports two basic operation modes: either a combined double-and-add step function or a single modular multiplication. The access to the modular multiplication instruction is required for the final inversion based on Fermat's little theorem, i.e, inversion of a field element $a \in \mathbb{F}_p$ by computing $a^{p-2} \bmod p$. To prevent timing attacks, the arithmetic unit performs the point multiplication running a total 255 double-and-add operations and 266 iterated multiplications for the inversion both in constant time.

In our implementation we follow several of the design suggestions for software implementations as given in the original work on Diffie-Hellman computations over Curve25519 [2]. In particular, each addition or subtraction is always followed by a subsequent multiplication again nearly always succeeded by another addition or subtraction. These facts led to the design presented in Figure 2 using two dual-ported BRAMs in butterfly configuration. More precisely, the first BRAM only receives the results of the addition or subtraction unit and provides the input to the multiplication while the second BRAM stores the multiplication result and feeds the addition unit. This way parallel operation is enabled and

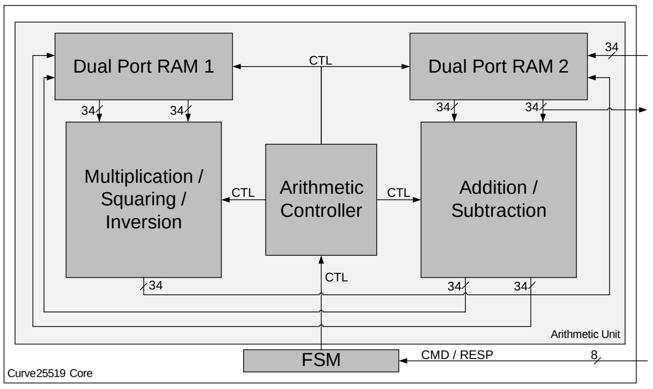

Fig. 2. Overview of the Curve25519 Core

pipeline stalls through loading and write-back can be avoided with only little overhead.

Modular Addition Unit Centerpiece of the modular addition and subtraction unit computing $c = a \pm b \bmod p$ are two DSP blocks supporting 25x18-bit multiplications and up to 48-bit additions, subtractions or accumulations. The first DSP always performs the main operation (i.e., subtraction or addition $c' = a \pm b$) whereas the second DSP block computes a prediction for a reduced result by $c'' = c' \mp p$. Both, the c' and c'' are stored into the first BRAM and distinguished by a flag obtained from the previous carry/borrow in the prediction operations that indicates in which registers the correct result is stored. In total, modular addition or subtraction takes 10 clock cycles which can be executed in parallel to any multiplication operation. Thus, exploiting the alternating operation flow as mentioned above, the latency for modular addition or subtraction is completely absorbed in the latency for a concurrently running modular multiplication.

Modular Multiplication Unit The largest component of the arithmetic unit is the multiplication unit and based on 18 DSP blocks – 15 blocks are used to compute partial products, one for a prereduction and two for the final modular reduction. A modular multiplication can be computed in 55 clock cycles of which 34 cycles are required for the actual multiplication and the remaining ones for loading and storing data. Due to the modular design shown in Fig. 3, computation of partials products (stage 1) can be interleaved with the reduction step (stage 2) in pipeline fashion. So a next multiplication operation can be already restarted as soon the first stage (partial products) has completed the previous multiplication Thus, only the first multiplication takes the full 55 clock cycles, each subsequent multiplication is becoming available with a latency of 17 clock cycles only. Since data dependencies need also been taken into account, the combined double-and-add step for Curve25519 finally takes 255 cycles in total.

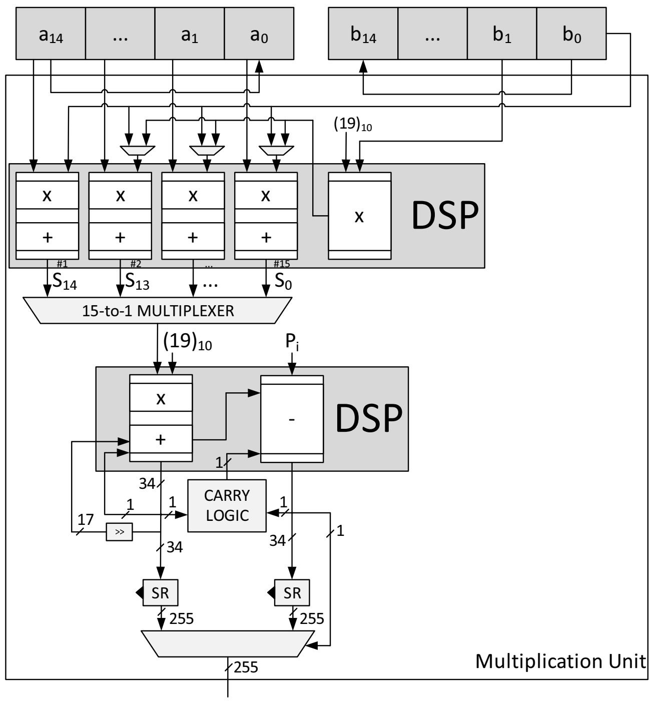

Fig. 3. Architecture of the Modular Multiplication Unit

5.2 Multi-Core Architecture

A main caveat with the single-core architecture is the slow inversion. In this work we augment the previously described core design with a dedicated inverter circuit and share it among several cores for an optimal cost-performance trade-off. The number of cores per inverter is upper-bounded by the available resources on the respective device as well as the relation of the cycle count per point multiplication with respect to one final inversion. Since this number directly corresponds to resources available on a given FPGA, we implemented the design generically to allow maximum scalability and flexibility also for other devices. Figure 4 shows the communication interface and the additional controller for distributing incoming packets among the Curve25519 cores. Unlike the hardcore introduced above, all cores of this architecture only support the step function (double-and-add operation) but no modular inversion anymore. The inversion is finally performed by a dedicated inversion unit shared by all cores in a subsequent step.

Dedicated Inversion Unit In many cases the modular division is the most expensive operation requiring a modular inversion and at least one multiplication. In the single-core approach, we noticed that inversion based on Fermat's

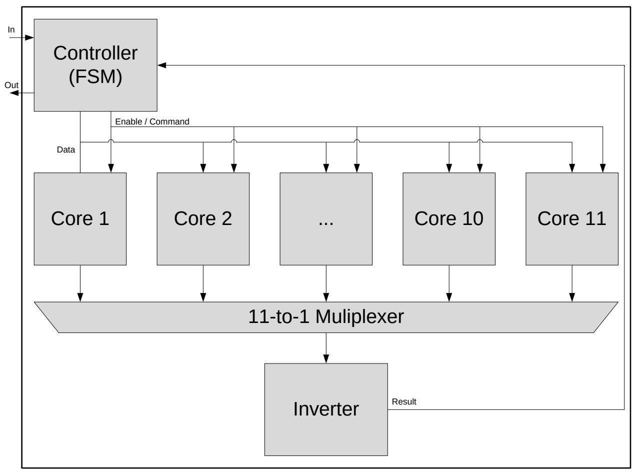

Fig. 4. Overview of the Multi-Core Design

little Theorem is rather ineffective since the inversion requires roughly 20% of the entire time of a Curve25519 computation. Therefore we implemented a modular inverter based on the Binary Extended Euclidean Algorithm as an extension to the existing cores. The inverter uses wide adders/subtracters and uses well-known implementation techniques so that we refrain from giving all details on the implementation. With respect to the multi-core design approach, the inverter receives X and Z as inputs, and computes the final result X/Z in a constant time. Since this inverter is significantly faster compared to a point multiplication, the inverter can serve as subsequent inversion unit for several point multiplication cores (cf. Section 6).

Load Balancing Due to the concurrent operation of individual cores, we implemented a scheme to distribute incoming data to available cores when they become available. This scheme is basically a round-robin-based loading scheme where the controller unit keeps track of the last active and the next available core. Loading continues until all cores are busy. As soon as one core reports a result, it is handed over to the inversion unit and the core is marked ready again and can be loaded with a next parameter set.

Clock Domain Separation The point multiplication cores can operate at a clock frequency of 200MHz. However, since the dedicated inversion unit implements 256-bit wide adders and subtracters in logic it only supports a maximum clock frequency of roughly 130MHz. To still operate the implementation at maximum speed, we use different clock domains for the point multiplication cores on the one hand (200 MHz) and the controller and inversion unit on the other hand (100 MHz).

Table 1. Summary of device utilization and performance

	Component	Used	Available	Utilization
Single-Core	Number of Slice Registers	3592	106400	3%
	Number of Slice LUTs	2783	53200	5%
	Number of occupied Slices	1029	13300	7%
	Number of DSP48E1	20	220	9%
	Number of RAMB36E1	2	140	1%
	Cycles per Step Function	64770@200MHz		
	Cycles per Inversion	14630@200MHz		
	Total Clock Cycles	79400@200MHz		
Multi-Core	Number of Slice Registers	43875	106400	41%
	Number of Slice LUTs	34009	53200	63%
	Number of occupied Slices	11277	13300	84%
	Number of DSP48E1	220	220	100%
	Number of RAMB36E1	22	140	15%
	Cycles per Step Function	64770@200MHz		
	Cycles per Inversion	1667@100MHz		
	Total Clock Cycles	34052@100MHz		

6 Results

All results were obtained after place-and-route based on a Xilinx Zynq XC7Z020 using Xilinx ISE 14.5.

6.1 Comparison of the Single- and Multi-Core Architecture

In Table 1 we provide the resource consumption for our single-core and multi-core design, respectively.

The single-core approach only uses a small portion of the (relatively) small Xilinx device, i.e., 7% of the Slices and 9% of the DSP slices. The remaining device components are available for any other function or application that needs to be implemented.

Our single-core architecture has a maximum operation frequency of 200MHz and needs roughly 80,000 clock cycles to perform a Curve25519 operation of which about 20% of the clock cycles are required to compute the final inversion. All in all, the core can perform about 2500 point multiplications per second.

Our multi-core scenario is obviously trimmed for highest performance using a moderately large FPGA. Due to the clock domain separation, the multi-core architecture can compute a point multiplication in about 34,000 cycles at a frequency of 100MHz. In addition, up to 11 operations can be performed in parallel with an initial latency of 1667 clock cycles which is the time that is required for the inversion. This leads to a final throughput of more than 32,000 Curve25519 operations per second.

Table 2. Selected high-performance implementations of public-key cryptosystems

Scheme	Device	Implementation	Logic	Clock	OP/s
Single-Core	XC7Z020	255-bit GF($2^{255}-19$)	1029 LS/20 DSP	200 MHz	2531
Multi-Core	XC7Z020	255-bit GF($2^{255}-19$)	11277 LS/220 DSP	100 MHz	21686
ECC [7]	XC4VFX12-12	256-bit GF(p), NIST	1715 LS/32 DSP	490 MHz	2020
ECC [10]	XC2VP125-7	256-bit GF(p), any	15755 LS/256 MUL	39.5 MHz	260
ECC [5]	Intel Core i3	255-bit GF($2^{255}-19$)	64 bit	2.1 GHz	10810
ECC [5]	AMD FX-4170	255-bit GF($2^{255}-19$)	64 bit	4.2 GHz	14285
RSA[17]	XC4VFX12-10	1024-bit with DSP	3937 LS/17 DSP	400 MHz	584
RSA[16]	0.5 μm CMOS	1024-bit, unified	28,000 GE	64 MHz	21

6.2 Comparison to other work

Despite the fact that this is, to the best of our knowledge, the first implementation of Curve25519 in hardware and that there exist only few implementations in software, we intend to also compare to other relevant work. In Table 2 we list some other implementation results which implement ECC on a comparable security level to Curve25519. Note, however, that due to varying technology of different FPGA generations a fair comparison is actually not accurately possible. In [7] the standardized NIST P-256 curve was implemented using a similar approach on a Xilinx Virtex 4. The authors report that a design to perform point requires more resources at a performance of about 2,000 point multiplications per second. We also like to highlight software results which we obtained e.g. from ECRYPT's eBATS project. According to the reported results, an Intel Core i3 can perform about 10,810 point multiplications per second using the Curve25519 function and an AMD Bulldozer FX-4170, running at 4.2, GHz can compute 14,285 Curve25519 operations per second.

Although comparing software and hardware can be misleading, we still like to emphasize that the moderately large and rather cheap Zynq 7020 FPGA can outperform some recent high-performance processors such as AMD Bulldozer and Intels Core i3. This is slightly surprising since elliptic curve cryptography over prime fields is usually considered a field that is dominated by software implementations with respect to pure performance.

7 Conclusion

In this paper we propose two new architectures for Elliptic Curve Cryptography using the Curve25519 supporting Diffie-Hellman key agreement and other cryptographic primitives. Both architectures provide a security level comparable to AES-128 and process data at constant time. We showed that the design can outperform many recently presented works in hardware and software using only moderate resource requirements.

References

1. ANSI X9.62-2005. American National Standard X9.62: The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical report, Accredited Standards Committee X9, `http://www.x9.org`, 2005.
2. Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.
3. Guerric Meurice de Dormale and Jean-Jacques Quisquater. High-speed hardware implementations of elliptic curve cryptography: A survey. *J. Syst. Archit.*, 53(2-3):72–84, 2007.
4. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22:644–654, 1976.
5. ECRYPT. eBATS: ECRYPT Benchmarking of Asymmetric Systems. Technical report, March 2007. `http://www.ecrypt.eu.org/ebats/`.
6. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31:469–472, 1985.
7. Tim Güneysu and Christof Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 62–78. Springer-Verlag, 2008.
8. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.
9. A.K. Lenstra and E.R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14(4):255–293, 2001.
10. C. McIvor, M. McLoone, and J. McCanny. An FPGA elliptic curve cryptographic accelerator over GF(p). In *Irish Signals and Systems Conference (ISSC)*, pages 589–594, 2004.
11. V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology — CRYPTO '85*, volume LNCS 218, pages 417–426, Berlin, Germany, 1986. Springer-Verlag.
12. Peter L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
13. G. Orlando and C. Paar. A scalable GF(p) elliptic curve processor architecture for programmable hardware. In *Cryptographic Hardware and Embedded Systems (CHES)*, volume LNCS 2162, pages 356–371, 2001.
14. S.B. Örs, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of elliptic curve processor over $GF(p)$. pages 433–443, 2003.
15. Kazuo Sakiyama, Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems. In Koen Bertels, João M. P. Cardoso, and Stamatis Vassiliadis, editors, *ARC*, volume 3985 of *Lecture Notes in Computer Science*, pages 347–357. Springer, 2006.
16. E. Savas, A.F.Tenca, M.E. Ciftcibasi, and C.K. Koc. Multiplier architectures for GF(p) and GF(2^n). *IEE Proc Comput Digit Tech*, 151(2):147–160, 2004.
17. Daisuke Suzuki. How to maximize the potential of FPGA Resources for Modular Exponentiation. In *Cryptographic Hardware and Embedded Systems - CHES*, volume 4727, pages 272–288, 2007.