A

Implementing Curve25519 for Side-Channel-Protected Elliptic Curve Cryptography

PASCAL SASDRICH, Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany
TIM GÜNEYSU, Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany

For security-critical embedded applications Elliptic Curve Cryptography (ECC) has become the predominant cryptographic system for efficient key agreement and digital signatures. However, ECC still involves complex modular arithmetic that is a particular burden for small processors. In this context, Bernstein proposed the highly efficient ECC instance Curve25519 that particularly enables efficient software implementations at a security level comparable to AES-128 with inherent resistance to simple power analysis (SPA) and timing attacks. In this work we show that Curve25519 is likewise competitive on FPGAs even when countermeasures to thwart side-channel power analysis are included. Our basic multi-core DSP-based architectures achieves a maximal performance of more than 32,000 point multiplications per second on a Xilinx Zynq 7020 FPGA. Including a mix of side-channel countermeasures to impede simple and differential power analysis we still achieve more than 27,500 point multiplications per second with a moderate increase in logic resources.

Categories and Subject Descriptors: B.2.4 [**Arithmetic and Logic Structures**]: High-Speed Arithmetic—Algorithms; B.5.1 [**Register-Transfer-Level Implementationg**]: Design—Arithmetic and logic units; C.1.1 [**Processor Architecture**]: Other Architecture Styles—ECC Co-Processor; E.3 [**Data Encryption**]: Public key cryptosystems

General Terms: Security

Additional Key Words and Phrases: ECC, Curve25519, Diffie-Hellman, Side-Channel Attacks, Zynq

1. INTRODUCTION

Many applications in the emerging Internet of Things (IoT) include complex public-key cryptography as a building block in their security system. In particular, the small and embedded computing devices in the IoT often need to provide a significant amount of cryptographic operations per second regardless the computational constraints of their processors. Since modern public key cryptosystems, e.g., RSA, use computation-intensive calculations with thousands of bits, these operations are often too slow. The proposal of Elliptic Curve Cryptography (ECC) [Miller 1986; Koblitz 1987] has enabled to achieve similar security compared to RSA but using smaller keys and with significantly less complex operations. 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), still providing an equivalent level of security [Lenstra and Verheul 2001]. Therefore ECC has emerged to the standard for high-performance asymmetric cryptography, although it still puts high requirements on small microprocessors and microcontrollers. Hence, flexible cryptographic hardware such as Field-Programmable Gate Arrays (FPGA) are still a preferred choice when high-performance of cryptographic operations is a requirement for such embedded application.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© YYYY ACM. 1936-7406/YYYY/01-ARTA $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

Contribution: In this work, we present a secure and efficient implementation of a special Elliptic Curve Cryptosystem using the Curve25519 [Bernstein 2006] on reconfigurable hardware. Our target application is the Diffie-Hellman key agreement suitable for high-performance applications. In particular, we extend our initial work in [Sasdrich and Güneysu 2014] to provide high-performance cryptography with physical protection against timing, simple power (SPA) and differential power (DPA) attacks. Our design takes particular advantage of the arithmetic hardcores for digital signal processing (DSP) of reconfigurable devices. Although Curve25519 was initially proposed to accelerate the Diffie-Hellman key agreement primarily in software, we show that its characteristics can be similarly exploited in hardware to achieve a compact and 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 and still more than 27,500 operations per second when enhanced with DPA countermeasures. With these results we can virtually support any high-performance application of asymmetric cryptography, even with high demands on the physical security of the implementation.

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 as well as side-channel based implementation attacks. 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 enhaced by side-channel countermeasures as described in Section 6. Finally, we compare our implementations in Section 7 and conclude our work in Section 8.

2. PREVIOUS WORK

We briefly summarize previously published results of relevance to this contribution. Since there is a wealth of publications addressing ECC hardware architectures and side-channel countermeasures for ECC implementations, we refer to the overview in [de Dormale and Quisquater 2007] and [Fan and Verbauwhede 2012] and restrict the discussion of previous work to the most relevant ones.

As one of the first works in ECC implementations, Orlando and Paar [Orlando and Paar 2001] proposed a design targeting explicitly reconfigurable hardware using Montgomery-based multiplications included with a series of precomputations. This publication was followed by many more, e.g., [Güneysu and Paar 2008] trying to improve performance on FPGAs by the use of dedicated multipliers or [Sakiyama et al. 2006] trying to improve the performance by an algorithmic approach. Using integrated Digital Signal Processors (DSP) both for modular multiplication and modular addition was initially proposed in [Ors et al. 2003] targeting standardized NIST primes P-224 and P-256 using a special reduction scheme.

Efficient computation on constrained embedded devices is only one side of the coin, but still achieving physical security against side-channel attack is the other. One of the first DPA attacks on elliptic curve hardware implementations was presented in [De Mulder et al. 2007]. To prevent DPA attacks, [Coron 1999] presented and implemented a set of countermeasures, including skalar and projective coordinate randomization, for a design performing Diffie-Hellman and ElGamal operations on elliptic curves.

3. BACKGROUND

In this section we will briefly introduce the mathematical background relevant to this work. We start with a short review of the Elliptic Curve Cryptosystems (ECC). Please

note that Curve25519 is defined over prime fields $GF(p)$. So all our background materials are restricted to those.

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. In particular, the use of projective coordinates avoids 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 $\mathcal{P}$. More precisely, the ECDLP is the fundamental cryptographic problem used in protocols and cryptographic schemes like the Elliptic Curve Diffie-Hellman key exchange [Diffie and Hellman 1976], the ElGamal encryption scheme [ElGamal 1985] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [ANSI X9.62-2005 2005]. 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 Specification

Curve25519 can be considered as a function designed to 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 [Montgomery 1987] 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 coordinates of the two points Q and Q'. Eventually, a combined

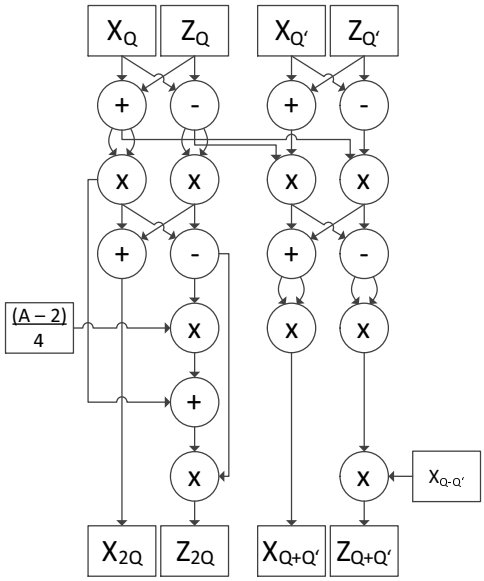

Fig. 1. Double-and-Add formula according to Montgomery's ladder.

step is computed by

$$\begin{aligned} x_{2Q} &= (x^2 - z^2)^2 = (x - z)^2(x + z)^2 \\ z_{2Q} &= 4xz(u^2 + Axz + z^2) \\ x_{Q+Q'} &= 4(xx' - zz') \\ z_{Q+Q'} &= 4(xz' - zx')x_1 \end{aligned}$$

3.2. Group Operations

For scalar multiplication $k \times P$ on the Curve25519, a total of 255 combined point double and addition operations 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 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, the number of operations performed is always the same, independently of the processed data. Therefore, the computation can be executed in constant time preventing timing-based attacks.

3.3. Timing Attacks

Timing attacks try to find dependencies between the secret stored on a device and the operation time of the cryptographic computations. For instance, the double-and-add algorithm used for many cryptosystems based on elliptic curve cryptography has such data dependencies if not implemented carefully. Due to the fact, that the algorithm decides for each bit of the secret scalar if a point doubling with or without subsequent point addition is performed, this will lead to a longer execution time if the bit pattern of the secret implies many combined point doubling & additions. By implementing algorithms that have a constant timing and therefore do not have these dependencies

(such as Montgomery's ladder used in the Curve25519 function), we can easily prevent timing attacks as such.

3.4. Simple Power Analysis

Power analysis is a powerful side-channel attack and was first (academically) discovered in [Kocher et al. 1999]. The basic idea of power analysis is to reveal a secret by measuring and inspecting the instantaneous power consumption of a device during its operation depending on the secret. For simple power analysis (SPA) we assume that we are only able to measure the instantaneous power consumption once or very few times. By visual inspection of the recorded traces, an attacker tries to find some noticeable patterns in the power consumption which correspond to some internal processes of the cryptographic device. If these operations are depending on the private key (or parts of it), the attacker can use these side-channel information to extract individual secret bits, finally recovering the full key.

Countermeasures for SPA aim to unify the patterns in the power traces to break any dependency on the secret. For instance, if a design implements the double-and-add algorithm, similarly to timing attacks an attacker could distinguish combined point double-and-add operations by different sequence lengths of patterns in the power trace. To achieve resistance against SPA, designers would implement algorithms that always perform an addition and a doubling, such as the Montgomery's Ladder of the Curve25519 function.

3.5. Differential Power Analysis

As simple power analysis, the differential power analysis (DPA) tries to exploit data or key dependencies in the power consumption. For the SPA this was done by visual inspection but for DPA one has to predict the power consumption of the device for a hypothetical value of the secret (or some part of it) and compare this hypothetical consumption to the measured consumption by applying statistical tests. Since statistical tools usually require large amount of traces, an attacker needs to record a high number of measurements, including a large amount memory and computational power. Based on a Divide-and-Conquer approach, usually small parts of the secret are guessed at a time which serve as input (along with some other known values) for the prediction of the data processing inside the cryptographic algorithm.

Common countermeasures to counteract DPA attacks try either to randomize the power consumption during computation or decorrelate it by equalizing power consumption for all data points. Equalization of the power consumption is hard to achieve, in particular on FPGAs with a given, fixed architecture. Therefore, randomizing the power consumption is typically preferable for reconfigurable devices [Mangard et al. 2008].

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 $c = 19$.

The given prime with a total of 255 bits can be divided into fifteen words of 17 bit width. Processing these chunks is usually inefficient for common processors which operate 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 combine 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 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 back 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 make use of Fermat's Little Theorem. This approach requires solely the modular multiplier already provided by the core that is augmented with a small additional state machine. Inversion based on this approach requires negligible extra resources, however, inversion performance will be comparably slow. Another approach would be to implement a dedicated inversion unit based on the binary Extended Euclidean Algorithm (EEA). This extra circuit requires a significant amount of additional resources but the computation of the inversion is significantly faster. The hardware costs can be mitigated in a multi-core scenario by sharing one dedicated inverter among several point multiplication cores.

Finally, we aim 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 first present 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 to provide maximum throughput on the Zynq 7020 FPGA.

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 of performing 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.

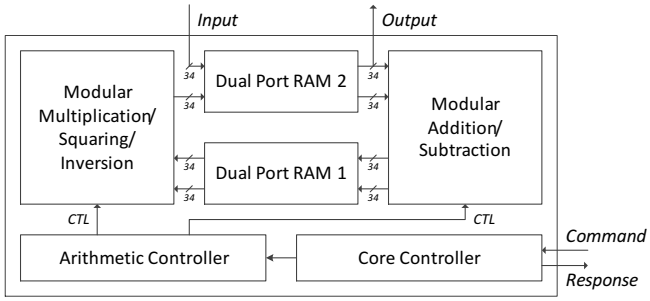

Fig. 2. Overview of the Curve25519 Core

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 [Bernstein 2006]. 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 pipeline stalls through loading and write-back can be avoided with only little overhead.

5.1.1. 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.

5.1.2. Modular Multiplication Unit. The largest component of the arithmetic unit is the multiplication unit and consists of 18 DSP blocks – 15 blocks are used to compute partial products, one for a pre-reduction 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 partial 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 takes 255 cycles in total.

5.2. Multi-Core Architecture

A main caveat of the single-core architecture is the slow inversion. In this work we augment the previously described core design with a dedicated inverter circuit and

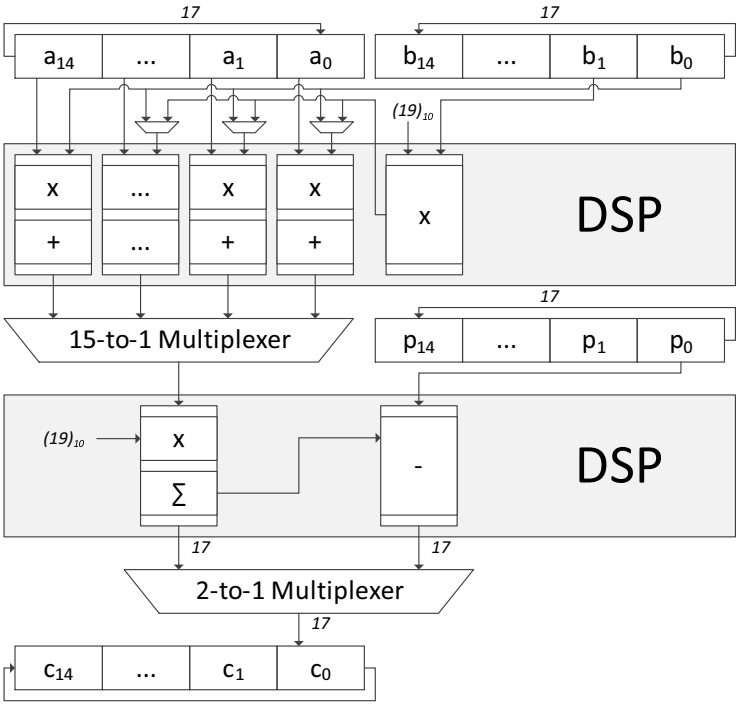

Fig. 3. Architecture of the Modular Multiplication Unit

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.

5.2.1. 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 little Theorem is rather costly since the inversion requires roughly 20% of the runtime of a full Curve25519 computation. Therefore, we opted to implement a modular inverter based on the Binary Extended Euclidean Algorithm as an extension to the existing core. The inverter uses wide adders/subtracters and 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 7).

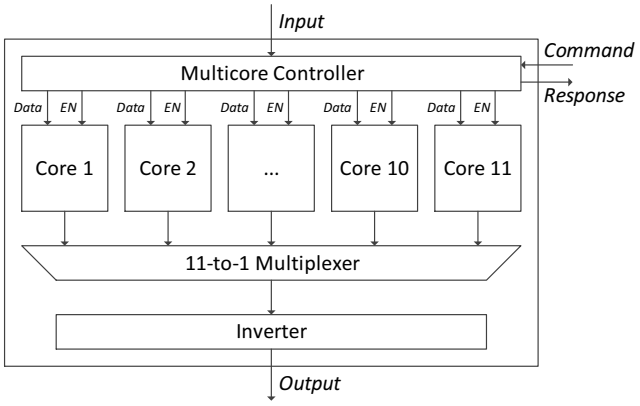

Fig. 4. Overview of the Multi-Core Design

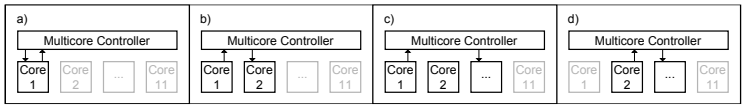

Fig. 5. Round-Robin-based Load Balancing Scheme: a) First core is loaded with data, marked busy and starts computing. b) Next data is handed over to Core 2 which is marked busy and starts operating, too. c) As long as data is available and cores are not busy, a new core is triggered with data. d) After finishing calculation, core 1 hands over the result to the inversion unit and is marked ready again.

5.2.2. 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 (cf. Figure 5). 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.

5.2.3. 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).

6. SIDE-CHANNEL COUNTERMEASURES

In [Coron 1999], several countermeasures were suggested to counteract DPA attacks on elliptic curve hardware implementations. Since our design is inherently resistant against timing and SPA attacks (cf. sections 3.3 and 3.4), we now present an extension of the Curve25519 core that adopts the proposed countermeasures against DPA attacks. In the following section we will briefly explain the idea of the countermeasures and describe their smooth integration into the core architecture.

6.1. Source of Randomness

In Section 3.5, we briefly mentioned two major approaches to counter differential power analysis: randomization (e.g. masking) or equalization of the power consumption. Since the equalization of the power consumption is a hard problem, especially

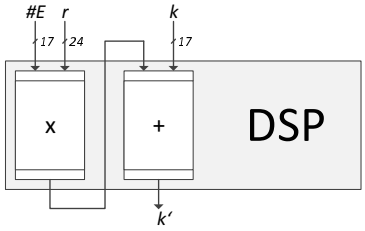

Fig. 6. Concept of the random scalar blinding countermeasure.

for FPGAs with their fixed routing and logic structure, we will focus our countermeasure approaches to randomization techniques. Therefore our countermeasures require a random number generator delivering the core with 285 bits of fresh randomness during every operation (255 bits for the random value λ, 24 bits for the random value r and 6 bits for the address randomization). Since the focus of this work is the implementation and integration of side-channel countermeasures into our Curve25519 design and not the design of random number generators, we assume that the randomness is provided by an external true or pseudo random number generator, e.g. based on a stream or block cipher.

6.2. Scalar Blinding

Most SPA and DPA attacks on elliptic curve implementations aim to reveal the secret scalar used in point multiplication. In order to thwart these attacks designers need to decorrelate the signatures of computations that involve the secret scalar from the power consumption.

The point multiplication in elliptic curve cryptography depends on two inputs, a public point P and a secret scalar k. Both can be target for randomization. The first approach randomly recomputes the public point in a way that intermediate results differ but the final result is still correct. This can be similarly applied to the private scalar by adding random multiples of the group order to it. More precisely, using a scalar $k*$ corresponding to the group order $\#E$ of the elliptic curve then the point multiplication $k * \times P$ always results in the point of infinity $\mathcal{O}$ which is the neutral element for the point addition. Hence, if we choose a scalar k' in a way that

$$k' = k + r \times \#E$$

then it holds for the point multiplication of the new scalar k' and a point P that

$$k' \times P = (k + r \times \#E) \times P = k \times P + r \times \mathcal{O} = k \times P$$

In [Coron 1999], a bitsize of at least 20 bits is suggested as sufficiently secure for the random value r. By using a DSP block for accelerating computation of k' we can even pick up to 24 random bits for r to serially compute the multiplication and final addition with the original k within 17 clock cycles. The final result k' has a maximum bit size of 280 bit what finally leads to 25 additional step function calls. Note that the entire computation remains still constant in time.

6.3. Randomized Projective Coordinates

Besides predicting parts of the random scalar, an attacker could try to guess parts of the binary representation of elliptic points and some intermediate values of the Double-and-Add formula in order to deduce the used scalar (even if its randomized). Internally, the affine coordinates (X, Y) of elliptic points are replaced by projective

coordinates (X, Y, Z) during computation whereby the computations for Curve25519 only depends on the coordinates (X, Z) and initially assumes $Z = 1$. This representation usually is used to relaxe the computation and replace costly inversions of the computation by additional multiplications. To protect against attacks on guessing the elliptic points, [Coron 1999] proposes a randomization of the projective representation which can be applied easily for our design. In order to randomize the point without affecting the result, we choose to multiply all coordinates with a random 255-bit λ during the initialization phase of the core. We therefore use

$$\lambda P = (\lambda X, \lambda Z) = (\lambda X, \lambda)$$

obtained from multiplication in our core so that the original coordinates (X, Z) stored in the BRAM are replaced with new coordinates $(\lambda X, \lambda)$. Note that all other values and constants of the computation are unaffected. The value λ is randomized for each execution so that, even when always using the same scalar and curve point as input to the computation, all intermediate results differ during execution of the point multiplication but still lead to the correct result.

6.4. Memory Address Scrambling

In order to protect the Curve25519 against timing and SPA attacks, Bernstein suggested to use the Double-and-And formula based on Montgomery's ladder presented in Section 3.2. While this formula provides security against timing and SPA attacks, a careless implementation can make the design vulnerable to DPA attacks.

For our implementation, the inputs of the algorithm, Q and Q' are stored in a BRAM and after the execution the results $2Q$ and $Q + Q'$ will overwrite Q and Q' and serve as a new input to the next round of the algorithm execution. For every execution of the algorithm, a single bit of the secret scalar is evaluated and indicates whether the inputs have to be swapped or not. Thus, the accessing of the BRAM during every execution of the Double-and-Add formula is key-dependent and can help an attacker to reveal the secret. Therefore the attacker simply has to reveal the BRAM addresses using a DPA. If the addresses of every execution would be known, the attacker then can identify the inputs of the algorithm and thereby the bits of the secret scalar.

To secure the memory accesses against side-channel attacks, the addresses should be randomized for each execution. For our design, we now choose random addresses for the intermediate values and all results that are stored in the memory during the point multiplication. After every execution of the point multiplication we then choose a new set of random addresses. Thus, we choose random addresses for the intermediate values and results that are stored in the BRAM and change them after every point multiplication.

Since the addresses of our BRAM have a size of 6 bits, we can select 2^6 different random addresses. With the help of a linear feedback shift register (LFSR) based on the formula $x^6 + x^5 + 1$, we can determine all addresses depending on a random start address. Therefore, we take 6 bits of the external randomness and use them as seed for the LFSR. Before the core can start computing a point multiplication, the LFSR is advanced and the random addresses are saved for the next multiplication, initializing the BRAM with the required values and constants.

In total we allow 2^6 different sets of random addresses instead of a single fixed one. Besides other technical issues such as noise reduction, and attack needs to learn how to distinguish between these memory lines. Since this classification problem is highly complex to solve when other countermeasure are enabled, we do not expect a DPA attack to be successful at any point.

	Component	Unprotected	Protected	Available	Utilization
Single-Core	Number of Slice Registers	3592	3784	106400	3% / 3%
	Number of Slice LUTs	2783	2862	53200	5% / 5%
	Number of Occupied Slices	1029	1180	13300	7% / 8%
	Number of DSP48E1	20	22	220	9% / 10%
	Number of RAMB36E1	2	2	140	1% / 1%
	Cycles per Step Function	64770	68880	*at 200MHz*	
	Cycles per Inversion	14630	14372	*at 200MHz*	
	Total Cycles	79400	83252	*at 200MHz*	
Multi-Core	Number of Slice Registers	43875	39916	106400	41% / 37%
	Number of Slice LUTs	34009	30582	53200	63% / 57%
	Number of occupied Slices	11277	10777	13300	84% / 81%
	Number of DSP48E1	220	220	220	100% / 100%
	Number of RAMB36E1	22	20	140	15% / 14%
	Cycles per Step Function	64770 @11 cores	68880 @10 cores	*at 200MHz*	
	Cycles per Inversion	1667	1667	*at 100MHz*	
	Total Cycles	34052	36107	*at 100MHz*	

7. RESULTS

All results were obtained after place-and-route based on a Xilinx Zynq XC7Z020 using Xilinx ISE 14.7.

7.1. Comparison of the Single- and Multi-Core Architecture

In Table 7.1 we provide the resource consumption for both the first and the countermeasure enhanced version for our single-core and multi-core design, respectively.

The single-core implementation requires only few resources on the Xilinx Zynq 7020 device, i.e. 7% (8%) of the Slices and 9% (10%) of the DSP blocks for the (protected) design. The remaining device components are available for any other function or application that are required to be implemented.

Both single-core designs, that means the protected and unprotected solution, have a maximum operation frequency of 200MHz. While the unprotected single-core needs 254 clock cycles to compute a single step function and 14630 cycles for the final inversion, this could even be decreased for the protected alternative by a slightly optimized modular multiplication unit. A step function now requires only 246 cycles and the final inversion only 14372. But due to the implementation of the additional countermeasures, some delay during the computation is added (cf. Table 7.1) which leads to about 83,000 for a total point multiplication instead of roughly 80,000. Note that still about 20% of the clock cylces are required to compute the final inversion.

Due to the implementation of additional countermeasures, our protected single-core has a slightly higher demand of resources, in particular on registers and DSP48E1. The most resource-consuming countermeasure is the blinding of the secret scalar, as one can see in Table 7.1. But based on the optimization of the single-core, the resource difference between the protected and unprotected alternative is much smaller than the overhead of the countermeasures.

All in all, the cores, protected and unprotected, can perform about 2,500 point multiplications per second while utilizing less than 10% of the FPGA.

Our multi-core implementation is optimized for high performance still targeting a moderately-large FPGA and making use of all resources available. Due to the clock domain separation, the multi-core architecture can compute a point multiplication in about 34,000-36,000 cycles at a frequency of 100MHz. In addition, up to 11 operations (or 10 using the cores enhanced with side-channel countermeasures) can be performed

Countermeasure	Delay		Area		
	Initialization	*Computation*	*LUTs*	*Registers*	*DSP*
Random Scalar Blinding	17	6150	306	311	2
Random Projective Coordinates	45	-	5	4	-
Random BRAM Addresses	22	-	1	6	-

Scheme	Device	Implementation	Logic	Clock	OP/s
Single-Core[1]	XC7Z020	255-bit GF($2^{255}-19$)	1029 LS/20 DSP	200 MHz	2519
Multi-Core[1]	XC7Z020	255-bit GF($2^{255}-19$)	11277 LS/220 DSP	11×100 MHz	32304
Single-Core[2]	XC7Z020	255-bit GF($2^{255}-19$)	1169 LS/22 DSP	200 MHz	2402
Multi-Core[2]	XC7Z020	255-bit GF($2^{255}-19$)	LS/220 DSP	10×100 MHz	27695
ECC[3]	XC4VFX12-12	256-bit GF(p), NIST	1715 LS/32 DSP	490 MHz	2020
ECC[4]	XC2VP125-7	256-bit GF(p), any	15755 LS/256 MUL	39.5 MHz	260
ECC[5]	XC4	256-bit GF(p), any	4655/37 DSP	291 MHz	2631
ECC[5]	XC5	256-bit GF(p), any	1725/37 DSP	250 MHz	2272
ECC[6]	Intel Xeon E3	255-bit GF($2^{255}-19$)	64 bit	4×3.5 GHz	37106
ECC[6]	AMD FX-8120	255-bit GF($2^{255}-19$)	64 bit	4×3.1 GHz	15567
ECC[6]	Cortex-A9+NEON	255-bit GF($2^{255}-19$)	64 bit	2×1.0 GHz	3460
ECC[6]	Snapdragon S4	255-bit GF($2^{255}-19$)	64 bit	2×1.5 GHz	8733

[1] unprotected, [2] protected, [3] [Güneysu and Paar 2008], [4] [McIvor et al. 2004], [5][Ma et al. 2013],[6] [ECRYPT 2007]

in parallel with an initial latency of 1667 clock cycles which is the time that is required for the inversion. For the side-channel protected scenario we can use only 10 cores in parallel because the limiting resource is still the number of DSP slices available and our scalar blinding countermeasure requires two additional DSPs for every core. But still, the multi-core approach leads to a final throughput of more than 32,000 (or 27,500 respectively) Curve25519 operations per second.

7.2. 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 extend the scope for our comparison to implementations over a different type of elliptic curves. In Table 7.2 we list implementation results for various platforms that 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 [Güneysu and Paar 2008] the standardized NIST P-256 curve was implemented using a similar approach on a Xilinx Virtex-4. The authors report their design to perform point requires more resources at a performance of about 2,000 point multiplications per second.

One more recent result, published in [Ma et al. 2013], implements a general ECC processor over GF(p) using DSP slices to enhance a Montgomery multiplier. The processor has been developed for Xilinx Virtex-4 and a newer Virtex-5 and can achieve a throughput of more that 2,200 operations per second. Note that this design also supports side-channel countermeasures while the designs in [Güneysu and Paar 2008] and [McIvor et al. 2004] are unprotected.

We also like to refer to software results which we obtained from ECRYPT's eBATS project. According to the reported results, an Intel Xeon E3 can perform about 37,000 point multiplications per second using the Curve25519 function and an AMD Bulldozer FX-8120, running at 3.1, GHz can compute 15,567 Curve25519 operations per second.

8. CONCLUSIONS

In this work we proposed two 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 to thwart timing attacks. We further included countermeasures to protect cryptographic operations against side-channel power analysis like SPA and DPA attacks with low resource overhead and a minor loss in performance. Finally we demonstrated that our designs are capable of outperforming many previous works both in hardware and software despite of their moderate resource requirements and countermeasures against implementation attacks.

Acknowledgment

This work was supported by the German Federal Ministry of Economics and Technology (Grant 01ME12025 SecMobil).

REFERENCES

ANSI X9.62-2005. 2005. *American National Standard X9.62: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. Technical Report. Accredited Standards Committee X9, http://www.x9.org.

Daniel J. Bernstein. 2006. Curve25519: New Diffie-Hellman Speed Records. In *Public Key Cryptography (Lecture Notes in Computer Science)*, Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin (Eds.), Vol. 3958. Springer, 207–228.

Jean-Sébastien Coron. 1999. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems*. Springer, 292–302.

Guerric Meurice de Dormale and Jean-Jacques Quisquater. 2007. High-speed hardware implementations of Elliptic Curve Cryptography: A survey. *J. Syst. Archit.* 53, 2-3 (2007), 72–84. DOI:http://dx.doi.org/10.1016/j.sysarc.2006.09.002

Elke De Mulder, Siddika Berna Örs, Bart Preneel, and Ingrid Verbauwhede. 2007. Differential power and electromagnetic attacks on a FPGA implementation of elliptic curve cryptosystems. *Computers & Electrical Engineering* 33, 5 (2007), 367–382.

W. Diffie and M. Hellman. 1976. New directions in cryptography. *IEEE Trans. Inf. Theory* 22 (1976), 644–654.

ECRYPT. 2007. *eBATS: ECRYPT Benchmarking of Asymmetric Systems*. Technical Report. http://www.ecrypt.eu.org/ebats/.

T. ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* 31 (1985), 469–472.

Junfeng Fan and Ingrid Verbauwhede. 2012. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In *Cryptography and Security: From Theory to Applications*. Springer, 265–282.

Tim Güneysu and Christof Paar. 2008. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems – CHES 2008 (Lecture Notes in Computer Science)*, Vol. 5154. Springer-Verlag, 62–78.

N. Koblitz. 1987. Elliptic Curve Cryptosystems. *Math. Comp.* 48 (1987), 203–209.

Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Advances in Cryptology — CRYPTO99*. Springer, 388–397.

A.K. Lenstra and E.R. Verheul. 2001. Selecting Cryptographic Key Sizes. *Journal of Cryptology* 14, 4 (2001), 255–293.

Yuan Ma, Zongbin Liu, Wuqiong Pan, and Jiwu Jing. 2013. A High-Speed Elliptic Curve Cryptographic Processor for Generic Curves over GF(p). In *Selected Areas in Cryptography*. 421–437.

Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2008. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer.

C. McIvor, M. McLoone, and J. McCanny. 2004. An FPGA elliptic curve cryptographic accelerator over GF(p). In *Irish Signals and Systems Conference (ISSC)*. 589–594.

V. Miller. 1986. Uses of Elliptic Curves in Cryptography. In *Advances in Cryptology — CRYPTO '85*, H. C. Williams (Ed.), Vol. LNCS 218. Springer-Verlag, Berlin, Germany, 417–426.

Peter L. Montgomery. 1987. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Math. Comp.* 48, 177 (1987), 243–264. DOI:http://dx.doi.org/10.2307/2007888

G. Orlando and C. Paar. 2001. A scalable GF(p) elliptic curve processor architecture for programmable hardware. In *Cryptographic Hardware and Embedded Systems (CHES)*, Vol. LNCS 2162. 356–371.

Sıddıka Berna Ors, Lejla Batina, Bart Preneel, and Joos Vandewalle. 2003. Hardware implementation of an elliptic curve processor over GF (p). In *Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on*. IEEE, 433–443.

Kazuo Sakiyama, Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. 2006. Reconfigurable Modular Arithmetic Logic Unit for High-Performance Public-Key Cryptosystems. In *ARC (Lecture Notes in Computer Science)*, Koen Bertels, João M. P. Cardoso, and Stamatis Vassiliadis (Eds.), Vol. 3985. Springer, 347–357.

Pascal Sasdrich and Tim Güneysu. 2014. Efficient Elliptic-Curve Cryptography Using Curve25519 on Reconfigurable Devices. In *Reconfigurable Computing: Architectures, Tools, and Applications*, Diana Goehringer, MarcoDomenico Santambrogio, JooM.P. Cardoso, and Koen Bertels (Eds.). Lecture Notes in Computer Science, Vol. 8405. Springer International Publishing, 25–36. DOI:http://dx.doi.org/10.1007/978-3-319-05960-0_3