Journal of Hardware and Systems Security (2018) 2:297–313
https://doi.org/10.1007/s41635-018-0048-z

Exploring RFC 7748 for Hardware Implementation: Curve25519 and Curve448 with Side-Channel Protection

Pascal Sasdrich[1] · **Tim Güneysu**[1]

Received: 14 February 2018 / Accepted: 24 August 2018 / Published online: 26 September 2018

Abstract
Recent revelations on manipulations and back-doors in modern ECC have initiated the revision of existing schemes and led to the selection of two new solutions for next-generation TLS proposed in RFC 7748: Curve25519 and Curve448. Unfortunately, both curves were designed and optimized primarily for software implementations; their implementation in hardware and physical protection against SCA has been neglected during the design phase. In this work, we demonstrate that both curves can indeed be efficiently and securely mapped to hardware structures of modern FPGAs while including advanced protection mechanisms against physical attacks and still providing high performance and throughput. In particular, our Curve25519 architecture provides more than 1 700 point multiplications per second, using only 1 006 logic slices (LSs) and 20 digital signal processors (DSPs) of a mid-range Xilinx XC7Z020 FPGA. Furthermore, our Curve448 architecture still achieves more than 600 operations per second at a significantly higher security level of 224 bits, using not more than 1 985 LSs and 33 DSPs on the same device. In addition, we performed a practical, test-based leakage assessment for both architectures. More precisely, we investigated the detection of scalar- and base-point-dependable leakage individually while our designs were incorporated *scalar blinding* and *point randomization* countermeasures. Eventually, our findings prove with high confidence, that we cannot detect any scalar- and base-point-dependable leakage even after evaluating 1 000 000 power measurements.

Keywords ECC · RFC7748 · TLS · Curve25519 · Curve448 · SCA · FPGA

1 Introduction

Efficient key agreement, exchange, and digital signatures are among the most crucial problems that cannot be solved solely relying on symmetric cryptographic primitives but require asymmetric public-key systems. For security-critical embedded applications, elliptic curve cryptography (ECC) has become the predominant asymmetric cryptographic system although it still involves complex modular arithmetic that is a particular burden for small embedded systems and processors.

However, recent revelations on manipulations and back-doors have undermined the confidence in existing schemes and led to discussion among researchers and standardization bodies on the selection of new curves and the rigidity of existing curve generation processes. Within these discussions, the Internet Engineering Task Force (IETF) Transport Layer Security (TLS) working group requested new recommendations on elliptic curves for the next generation of TLS on which the Internet Research Task Force (IRTF) Crypto Forum Research Group (CFRG) has selected two candidates for the RFC 7748: Curve25519 and Curve448. Simultaneously, the IETF *curdle* group proposed both curves for application in future protocols such as DNSSEC. With further emergence of the Internet of Things (IoT), TLS (including Curve25519 and Curve448) has to be implemented on various embedded and constrained devices. But since both curves were primarily designed for powerful software platforms (x86) and with hardly any evaluation of their resistance against low-level physical threats such as side-channel attacks, we still need to investigate the hardware implementation of both curves in combination with according countermeasures.

✉ Pascal Sasdrich
pascal.sasdrich@rub.de

Tim Güneysu
tim.gueneysu@rub.de

[1] Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Bochum, Germany

1.1 Related Work

Since there is a wealth of publications addressing ECC hardware architectures and side-channel countermeasures for ECC implementations that we cannot cover and discuss within the scope of this work, we refer an interested reader to the overviews in [5, 7, 8] and restrict the discussion of related work to the most relevant ones.

As one of the first attempts in hardware-based ECC implementations, Orlando and Paar [17] proposed a design targeting explicitly reconfigurable hardware using Montgomery-based multiplications including a series of precomputations. This publication was followed by many more, e.g., [9] trying to improve performance on field-programmable gate arrays (FPGAs) by using of dedicated multipliers or [21] trying to improve the performance by an algorithmic approach. Using integrated DSPs both for modular addition and multiplication was initially proposed in [18] targeting the standardized NIST primes P-224 and P-256 using a special reduction scheme. Besides, Järvinen et al. [11] recently presented a high-performance architecture for FOUR$\mathbb{Q}$, a novel elliptic curve with about 128-bit security that supports highly efficient point multiplications. Recently, Koppermann et al. presented a novel optimized architecture for X25519, explicitly targeting low-latency applications for the realm of IoT devices by heavily exploiting parallelisms in the Montgomery ladder algorithm.

However, efficient computation on constrained embedded devices is only one side of the coin, but still achieving physical security against side-channel analysis is the other. One of the first differential power analysis (DPA) attacks on ECC hardware implementations was presented in [16]. To prevent common DPA attacks, Coron proposed and implemented a set of countermeasures, including scalar and projective coordinate randomization, for a design performing Diffie-Hellman and ElGamal operations on elliptic curves [4]. In [2], a flexible hardware architecture for standard NIST primes p_{192}, p_{224}, p_{256}, p_{384}, and p_{521} was proposed that implements low-level and high-level protection mechanisms against simple power analysis (SPA) and DPA including point randomization and the unification of operation sequences to prevent adversaries from distinguishing atomic operations.

1.2 Contribution

In this work, we present side-channel protected, efficient, and high-performance architectures of special ECC systems implementing the particular elliptic curve instances CURVE25519 [3] and CURVE448 [10] on reconfigurable hardware. In general, the main target application of this work is a single point multiplication operation suitable for high-performance applications, i.a., necessary for Diffie-Hellman key agreement protocols and digital signature generation and verification. Particularly, we extend our initial works presented in [22–24] to provide practical, test-based leakage and side-channel evaluation of our high-performance cryptography architectures including physical protection against common side-channel analysis such as timing behavior, SPA, and DPA.

In general, our primary design objective was to prove that the mapping of both curves to (reconfigurable) hardware structures can be done efficiently. To this end, all our designs take particular advantage of the arithmetic cores intended for digital signal processing provided by most modern reconfigurable devices. Although CURVE25519 and CURVE448 were initially proposed to accelerate the Diffie-Hellman key agreement primarily in software, we show that the characteristics for both elliptic curves can be similarly exploited in hardware to achieve and implement a compact and high-performance ECC processor on modern reconfigurable devices.

Due to its small parameter sizes and efficient arithmetic, nowadays, ECC is considered for many modern high-performance applications with high demands on throughput and computational power. To this end, we strove for high-performance implementations in order to provide architectures that can provided sufficient computational power and acceleration for virtually every application. Depending on the provided security level, our goal was to perform at least 1 000 operations per second for the medium 128-bit security level of CURVE25519 and still 500 operations per second for high-security applications using CURVE448. Eventually, our single-core CURVE25519 architecture, implemented on a moderate Xilinx XC7Z020 FPGA, achieves more than 1 739 point multiplications per second. On the other hand, despite of the similarity of CURVE25519 and CURVE448, we were faced with completely revising the design rationales due to specially crafted parameters and the significantly increased field size of 448 bits. Interestingly, we can demonstrate that even in light of the asymptotically growing complexity in the field size, CURVE448 still can provide similarly high throughput providing a performance of more than 683 point multiplications per second at moderate resource costs.

Despite the security level provided by the hardness of the underlying mathematical problem, modern cryptographic systems more often have to face severe threats due to implementation attacks. In particular for modern embedded systems and devices, including various cryptographic implementations and co-processors, physical attacks such as side-channel analysis (SCA) and fault-injection analysis (FIA) are serious problems. Given physical access to a cryptographic system, the adversary may observe additional information such as timing [12], power consumption [13], or electromagnetic emanations [1]. Consequently, modern cryptographic implementations include a broad variety of countermeasures against these attack vectors. To this end, our implementations inherently provide protection against timing and SPA attacks, but we include advanced security mechanisms in order to provide even in the light of DPA attacks. Further, we present test-based leakage evaluation results using 1 000 000 power traces that confirm the security of our proposed architectures with high confidence.

Providing these results, our designs can virtually support any high-performance application of asymmetric cryptography using reconfigurable devices, even with high demands on the physical security of the implementation.

2 Preliminaries

In this section, we will discuss the basic field arithmetic of both considered elliptic curves (CURVE25519 and CURVE448) before presenting the fundamentals for the group arithmetic and the variable point-scalar multiplication operation. Furthermore, we include information on side-channel protection mechanism and discuss general optimization strategies for hardware implementation.

2.1 Field Arithmetic

2.1.1 Curve25519

The elliptic curve CURVE25519 is an efficient Montgomery curve specified over a prime field with a prime of shape $2^n - c$, i.e., close to a power of two (Pseudo Mersenne prime), and defined by its short Weierstrass equation:

$$\mathcal{E}_M : y^2 = x^3 + 486\,662x^2 + x \bmod (2^{255} - 19) \tag{1}$$

with $A = 486\,662$, $d = \frac{(A-2)}{4}$ and $p = 2^{255} - 19$. In general, according to RFC 7748, CURVE25519 provides the function `X25519` in order to accelerate the Diffie-Hellman key agreement, but also can be used for digital signature schemes based on its birationally equivalent Edwards curve ED25519.

On the lowest level, CURVE25519 processes 255-bit values and performs arithmetic operations over $GF(p)$ defined by $p = 2^{255} - 19$. In general, field arithmetic operations include modular addition, subtraction, multiplication, squaring, and inversion. Since the underlying finite field is defined over a Pseudo Mersenne prime, providing a special structure, efficient reduction can be performed using the fact that $2^{255} \equiv 19 \bmod p$. Eventually, inversion can be reduced to modular multiplication using Fermat's Little Theorem (FLT), i.e., $a^{p-2} \equiv a^{-1}$. Hence, given a set of modular addition, substraction, and multiplication operations and instructions, we can realize any functionality that is necessary in order to implement group arithmetic operations as well as the point multiplication.

2.1.2 Curve448

The untwisted Edwards curve CURVE448[1] with a security level of 224 bits is given and defined by:

$$\mathcal{E}_D : y^2 + x^2 \equiv 1 + dx^2y^2 \pmod{2^{448} - 2^{224} - 1} \tag{2}$$

with factor $d = -39\,081$, the Solinas prime $p = 2^{448} - 2^{224} - 1$ and its golden ratio $\phi = 2^{224}$. It can be used in order to accelerate variable point-scalar multiplications with an extended security level of 224 bits.

Technically, CURVE448 processes 448-bit values over $GF(p)$ but still is highly versatile and flexible for software implementations and different platforms (ranging from 8 to 64 bits) due to the fact that $448 = 56 \times 8 = 28 \times 16 = 14 \times 32 = 7 \times 64$. Further, due to its golden ratio ϕ, the Solinas prime p allows fast and efficient Karatsuba-based multiplication of two operands $A = (a_0 + a_1\phi)$ and $B = (b_0 + b_1\phi)$, such that:

$$\begin{aligned} &C = A \cdot B = (a_0 + a_1\phi) \cdot (b_0 + b_1\phi) = \\ &(a_0b_0 + a_1b_1) + ((a_0 + a_1) \cdot (b_0 + b_1) - a_0b_0)\phi \end{aligned} \tag{3}$$

Eventually, modular inversion can be tweaked compared to inversions based on FLT using that, if $p \equiv 3 \bmod 4$, the inverse square root (ISR) can be computed as $\frac{1}{\pm\sqrt{x}} = x^{\frac{(p-3)}{4}}$, which directly leads to:

$$x^{-1} = x \cdot \left(\frac{1}{\pm\sqrt{x^2}}\right)^2 = x \cdot [(x^2)^{\frac{(p-3)}{4}}]^2 = x^{p-2} \tag{4}$$

2.2 Group Arithmetic

According to Request for Comments (RFC) 7748, both functions `X25519` and `X448` perform a variable scalar-point multiplication on the according Montgomery curve (CURVE25519 or CURVE448) using the Montgomery ladder algorithm [15] that combines *point addition* and *point doubling* in a single step. Hence, given d, two points $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathcal{E}$ (in projective coordinate representation) and the difference $\mathcal{Q}_3 = \mathcal{Q}_1 - \mathcal{Q}_2$, a single step of the Montgomery ladder algorithm, computes two points $\mathcal{Q}_4, \mathcal{Q}_5 \in \mathcal{E}$ such that $\mathcal{Q}_4 = 2 \cdot \mathcal{Q}_1 = (x_4, z_4)$ is the result of the point doubling with:

$$x_4 = (x_1 - z_1)^2 \cdot (x_1 + z_1)^2 \tag{5}$$

$$z_4 = 4x_1z_1 \cdot (x_1^2 + dx_1z_1 + z_1^2) \tag{6}$$

[1]According to RFC 7748, CURVE448 represents a Montgomery curve and the untwisted Edwards curve actually is called Edwards448. However, since both curves are birationally equivalent, we use the term CURVE448 synonymously throughout this work.

and $\mathcal{Q}_5 = \mathcal{Q}_1 + \mathcal{Q}_2 = (x_5, z_5)$ is the result of the point addition with:

$$x_5 = z_3((x_1 - z_1)\cdot(x_2 + z_2) + (x_1 + z_1)\cdot(x_2 - z_2))^2 \quad (7)$$
$$z_5 = x_3((x_1 - z_1)\cdot(x_2 + z_2) - (x_1 + z_1)\cdot(x_2 - z_2))^2 \quad (8)$$

Fortunately, these equations are independent of the y-coordinate which can be omitted during the intermediate computation and only has to be restored for the final result using Eqs. 1 and 2.

Figure 1 shows the algorithmic flow of a single step for three points $\mathcal{Q}_1$, $\mathcal{Q}_2$, and $\mathcal{Q}_3$. In total, a single step of the Montgomery ladder algorithm involves 7 multiplications, 4 squarings, 4 additions, and 4 subtractions over $GF(p)$. The number of operations performed and its sequence is always the same, independently of the processed data. Therefore, the computation for both curves can be executed in constant time preventing timing-based attacks and even SPA.

2.3 Point Multiplication

Given a public point $\mathcal{P} \in \mathcal{E}$ and a secret k, the point multiplication routine computes another point $\mathcal{Q} = k \cdot \mathcal{P}$ using a sequence of consecutive Montgomery ladder steps. For this purpose, the scalar is scanned bit-wise (starting from the most significant bit) and depending on the value of the currently processed bit, inputs to a single step of the Montgomery ladder algorithm are swapped. Hence, starting from the base point $\mathcal{P}$, the point at infinity $\mathcal{O}$, and their difference (i.e., $\mathcal{P}$), the point multiplication finally yields the resulting point $\mathcal{Q}$ after the execution of 255 steps in the case of CURVE25519 and 448 steps for CURVE448.

2.4 Side-Channel Protection

In order to prevent SCA attacks, modern cryptographic implementations usually encompass protection and countermeasures against various physical attacks. Fortunately, the application of the Montgomery ladder algorithm already provides basic and inherent protection against timing and SPA attacks; however, dedicated countermeasures against DPA attacks have to be added. Since there is a plethora of different countermeasures against DPA attacks, we refer the interested reader to an overview of Fan et al. [7, 8], but only briefly summarize *point randomization* and *scalar blinding* as an example.[2]

2.4.1 Point Randomization

This countermeasure takes advantage of the additional degree of freedom introduced by the application of projective coordinate representations in order to randomize the representation of a base point using a randomly chosen λ. During the base point transformation process (from affine to projective coordinates), the random factor is applied as shown in Eq. 9, which yields in different point representations (depending on λ).

$$\mathcal{P}_r = (x_{\mathcal{P}_r}, z_{\mathcal{P}_r}) = (\lambda x_{\mathcal{P}}, \lambda z_{\mathcal{P}}) = (\lambda x_{\mathcal{P}}, \lambda) \quad (9)$$

Fortunately, *point randomization* does not affect the correctness of the underlying scheme, as proven in Eq. 10.

$$x_{\mathcal{P}_r} \cdot z_{\mathcal{P}_r}^{-1} = \lambda x_{\mathcal{P}} \cdot \lambda^{-1} = x_{\mathcal{P}} \pmod{p} \quad (10)$$

2.4.2 Scalar Blinding

Instead of randomizing the base point $\mathcal{P}$, *scalar blinding* deals with the randomization of the second input parameter of the *point multiplication* operation that computes $\mathcal{Q} = k \cdot \mathcal{P}$. Again, choosing a random factor r, the original scalar k is blinded with a multiple of the group order $|\mathcal{E}|$, as shown in Eq. 11.

$$k_r = k + r \cdot |\mathcal{E}| \quad (11)$$

Given the blinded scalar k_r, correctness of the underlying scheme is proven by the fact that the product of group order and base point returns the point at infinity. Hence, the *point multiplication* still results in $\mathcal{Q}$ as shown in Eq. 12.

$$k_r \cdot \mathcal{P} = (k + r \cdot |\mathcal{E}|) \cdot \mathcal{P} = k \cdot \mathcal{P} + r \cdot \mathcal{O} = k \cdot \mathcal{P} \quad (12)$$

Due to structures within the group orders, the randomization factor for the scalar blinding countermeasure has to be chosen appropriately. Hence, Schindler and Wiemers [25] suggest to choose blinding factors of more that half of the bit-length of the according group order which naturally increases the number of Montgomery steps per operation.

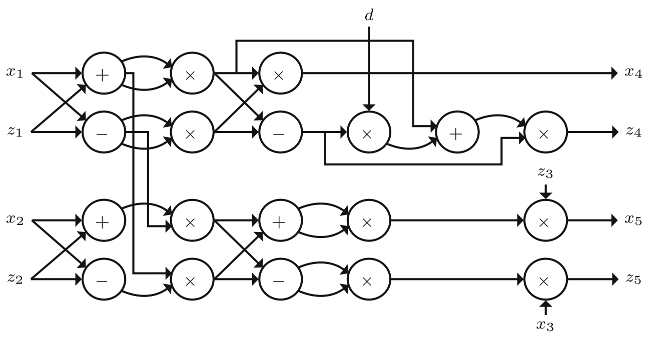


Fig. 1 Visualization of the combined double-and-add formula according to Montgomery's ladder

[2]We have chosen this set of countermeasures since it provides us with the opportunity to randomize and protect all input parameters of a single *point multiplication*.

2.5 Optimization Strategies for Reconfigurable Hardware

In order to provide the best possible mapping of the chosen elliptic curves and algorithms to the provided structures of modern reconfigurable devices, we heeded several optimization strategies. We identified these approaches during our design space exploration while evaluating different architectural choices in order to provide the optimal solution for the hardware structures of modern Xilinx FPGAs. In addition, we still followed generic and well-known optimization strategies including *register balancing*, *parallelization*, and *resource sharing* in order to increase the overall throughput of our hardware architectures. Hence, given a single instance of our design, we put a lot of engineering effort to find the optimal mapping on reconfigurable hardware, minimize the critical path, and increase the maximum frequency by hand while maintaining a balanced resource utilization in order to allow the implementation of multiple instances in parallel (multi-core design). In the following, we briefly explain algorithmic conditions for efficient mapping on reconfigurable hardware and provide some details on our optimization strategies and approaches.

Control Flow. Most algorithms that are implemented in hardware will be mapped to finite state machines (FSMs) in order to handle and control iterations and sequences of consecutive operations. In particular algorithms with a regular structure, few exceptions and repetitive sequences of operations will result in small FSMs with low or medium complexity. On the other hand, even larger algorithms that can be reduced to smaller sequences can help to reduce the overall complexity while implementing multiple FSMs that interact with each other (*divide-and-conquer* approach).

Parallelism. One of the major advantages of hardware implementations, including reconfigurable hardware, is the instantiating of multiple independent blocks that take care of individual parts of an algorithm which can be executed in parallel. In particular for reconfigurable devices, this is easy to realize since FPGAs provide a plethora of general purpose logic resources that can be configured in order to support virtually all possible tasks without additional costs.

Data and Memory. In addition, modern FPGAs provide special purpose logic blocks, such as DSPs and block-RAMs (BRAMs), each with a configurable interface in order to support different data widths. In particular, the BRAMs are powerful resources that can be used in order to store large amounts of data. Through the configurable interface, the data can be accessed in various ways and multiple BRAMs can be cascaded in order to extend the interfaces and memory space. Apart from that, LUT-based distributed memory (LUTRAM) can be used as small but fast alternative to store smaller portions of data more efficiently. Hence, even data-intensive algorithms with varying data widths can be mapped easily to reconfigurable hardware.

3 Side-Channel Protected Curve25519

In this section, we discuss the key features of CURVE25519 that enable an efficient map of its mathematical and arithmetic structures to contemporary reconfigurable hardware devices. In addition, we provide insights to the most relevant features of our side-channel protected, high-performance, single-core architecture and provide practical implementation results for mid-range Xilinx FPGAs.

3.1 Design Considerations

In the following, we identified the several aspects of CURVE25519 at different hierarchies as promising features that allow to map the necessary arithmetic efficiently on modern FPGA structures.

3.1.1 Field Arithmetic

On the lowest level, the field arithmetic operations provide ample scope for optimization and efficient mapping on reconfigurable structures.

Modular Reduction. For most modern standardized elliptic curves considered for high-performance applications, e.g., the NIST P-224 and P-256 instances, the underlying prime field is based on a Generalized Mersenne Prime which allows a modular reduction solely based on additions and subtractions. However, the underlying finite field of CURVE25519 is based on a Pseudo Mersenne Prime of shape $2^n - c$ providing a slightly different modular reduction scheme. In particular, for the elliptic curve instance CURVE25519, the reduction can be computed by a multiplication with the small constant $c = 19$ and additional additions or subtractions.

Modular Multiplication. Any field element, with a total size of 255 bits, can be divided perfectly into fifteen chunks, each having a size of 17 bits. Processing smaller words is usually inefficient for common processors and microcontrollers which often operate on a basis of 8-, 16-, 32-, or 64-bit data words. Modern FPGA devices, however, provide a multitude of dedicated, full-custom designed, arithmetic DSP slices equipped with pre-addition, multiplication, and accumulation or addition stages for fast integer arithmetic. Each DSP instance is enhanced with additional register stages to reduce the internal critical path delay and allow operations at maximum device frequency. Since the multiplication unit within each DSP block was designed to support signed 25×18-bit wide multiplications (or 24×17-bit unsigned operations), this is a perfect fit to our requirement of processing unsigned 17-bit data words. By means of a customized, interleaved multiplication scheme, multiplying two 255-bit field elements can be rearranged and distributed over several DSP blocks operating in parallel. Each single DSP unit then has to compute only one 17×17-bit multiplication at a time. Additionally, we can use the included accumulation stage within each DSP core to sum up the intermediate results into a partial product.

Modular Addition and Subtraction. Besides modular multiplication, the basic group operations (point doubling and addition) require the computation of modular additions and subtractions. Fortunately, the addition respectively subtraction unit can be implemented as simple cascade of only two DSP instances, one to perform the main arithmetic operation and one for the subsequent modular reduction.

Modular Inversion. In order to return a unique result for each point multiplication operation, a final inversion is required to convert the internal result based on projective coordinates back to an affine coordinate representation. For the basic version of a modular inversion, we make use of FLT. 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, its performance will be comparably slow. Another approach would be to implement a dedicated inversion unit based on the binary Extended Eucledean Algorithm (EEA). This extra circuit requires a significant amount of additional resources but the computation of each inversion is significantly faster. Still, in a multi-core scenario, these hardware costs can be mitigated mostly by sharing one dedicated inversion unit among several point multiplication cores.

3.1.2 Architectural Level

From a different point of view, modern FPGAs provide plenty of resources that allow implementation of custom and tailored architectures for virtually all modern applications. To this end, our architecture particularly draws upon BRAM and DSP cores:

Memory. We limit the storage requirements to a bare minimum, using only two 36K dual-port BRAM units to store all intermediate values in a butterfly-wise configuration and data flow using an enhanced 34-bit data path to process two words at the same time.

Arithmetic. As mentioned before, the addition and subtraction unit can be implemented with a minimal amount of two DSP cores. Besides, the full multiplication unit contains 15 DSP slices in parallel and we end up generating the final result in the accumulation stage that is slightly too large but which can be reduced in a subsequent recombination step. The final reduction 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 still slightly too large. Eventually, we can implement the entire field arithmetic operations (addition, subtraction, multiplication, squaring, inversion, and reduction) using only 20 DSP units.

3.2 Implementation

Our single-core CURVE25519 implementation is designed as a supplementary unit to provide asymmetric, public-key cryptographic operations, but to save most of the FPGA logic and resources for additional main applications. In detail, the cryptographic core is designed to support point multiplications using points on CURVE25519. More precisely, the single-core architecture supports addition and doubling operations of points given in projective coordinate representations. However, for use with most cryptographic protocols, the core additionally implements an inversion functionality to convert final results from projective back to affine coordinate representations. Internally, the arithmetic processor therefore supports two basic operation modes: either a combined double-and-add step function using the Montgomery ladder algorithm or a single modular multiplication operation. In particular, the latter instruction is required for the final conversion, i.e., for a modular inversion of a field element $a \in GF(p)$ by computing $a^{p-2} \bmod p$ based on FLT. To prevent timing attacks, the arithmetic unit performs the point multiplication running a total 384 double-and-add operations (including additional steps due to the scalar blinding countermeasure) and 266 iterated multiplications for the inversion, both in constant time.

In general, our implementation follows several of the design suggestions of Bernstein for software implementations as given in the original work on Diffie-Hellman computations over CURVE25519 [3]. In particular, each addition or subtraction is followed by a subsequent multiplication again nearly always succeeded by another addition or subtraction. This fact leads to a design using two dual-port BRAM units in a butterfly-like configuration. More precisely, the first BRAM only receives results of the addition or subtraction unit and provides inputs to the multiplication unit. Further on, the second BRAM stores all results of the multiplication core and feeds the addition unit. This way, parallel operation of both arithmetic units is enabled and pipeline stalls through loading and write-back can be avoided with only little overhead.

3.2.1 Modular Addition and Subtraction Unit

Centerpiece of the modular addition and subtraction unit (computing $c = a \pm b \bmod p$) are two DSP blocks supporting 25×18-bit signed multiplications and up to 48-bit additions, subtractions, or accumulations. Whereas the first DSP always performs the main operation (i.e., addition or subtraction $c' = a \pm b$), the second DSP block computes a prediction for the reduced result by $c'' = c' \mp p$. Both c' and c'' are stored into the first BRAM and distinguished by a flag which is obtained from the last carry or borrow bit in the prediction operation and indicates at which address 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, when exploiting the alternating, butterfly-like data flow as mentioned above, the latency for modular addition or subtraction is completely absorbed within the latency for a concurrently running modular multiplication.

3.2.2 Modular Multiplication Unit

Besides the modular addition and subtraction unit, the largest component of the arithmetic core is the modular multiplication unit which consists of 18 DSP instances — 15 blocks are used for partial product computations,

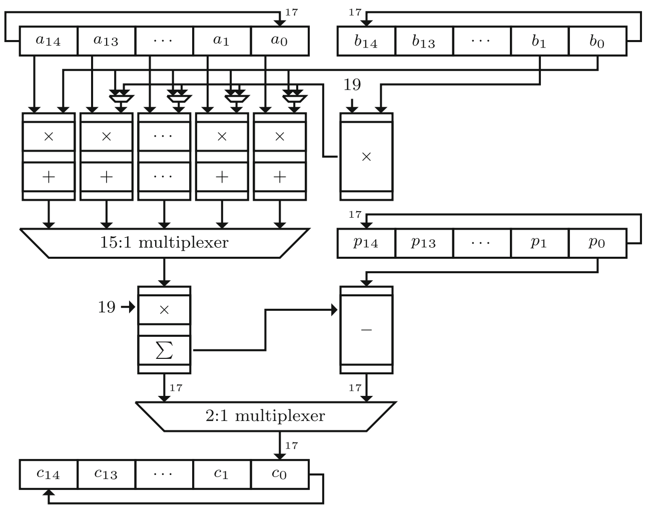


Fig. 2 Architecture of the modular multiplication unit

one for pre-reduction and two for the post-reduction and recombination step. Each modular multiplication can be computed in 55 clock cycles of which 34 are required for the actual multiplication and the remaining ones for loading and storing data. Due to the modular design shown in Fig. 2, computation of partial products (*stage 1*) can be interleaved with the reduction step (*stage 2*) in pipeline fashion. By this, new multiplication operations can be already restarted as soon as the first stage (computing the partial products) has completed the previous computation. Thus, only an initial multiplication takes the full 55 clock cycles, whereas each subsequent computations is becoming available with a latency of 17 clock cycles only. However, since data dependencies need also been taken into account, the combined double-and-add step for CURVE25519 takes 246 cycles in total.

3.2.3 Side-Channel Countermeasures

In Section 2.4, two countermeasures were presented that allow preventing DPA attacks on elliptic curve hardware implementations. Since our design is inherently resistant against timing and SPA attacks, we now briefly present the details of the smooth integration of side-channel countermeasures into the core architecture.

Source of Randomness Since most modern side-channel protection mechanism rely on randomization of internals, our countermeasures require an additional random number generator (RNG) delivering the core with at least 389 bits of fresh randomness during every operation. In particular, the core accepts points in projective coordinate representation that can be exploited to randomize the base point. However, the point randomization, i.e., the modular multiplication $\lambda x_{\mathcal{P}} \pmod p$, has to be applied externally. Similarly, the core accepts randomized scalars of up to 384 bits, that have been blinded beforehand using a 128-bit random blinding factor. Optionally, the core can be forced to randomize and scramble internal register addresses using another 6 bits of randomness.

Scalar Blinding Since the point multiplication in ECC depends on two parameters, a public point $\mathcal{P}$ and a secret scalar k, both can be target for randomization. In our first approach, the private scalar is randomized (externally) and blinded by adding random multiples of the group order of the underlying elliptic curve. Due to the special structure of the group order $\#\mathcal{E}$, it turns out that small blinding factors are insufficient to hide the structure which could be exploited using advanced SCA [25]. Thus, in order to prevent such attacks, practical solutions have to rely on blinding factors of at least half of the bit size of the group order, i.e., we considered 129 bits for CURVE25519. Apart from that, the core uses 384 instead of 255 Montgomery ladder steps to perform a full point multiplication operation, certainly affecting and decreasing the final throughput of our core.

Simple Point Randomization 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 operation in order to relax the computation and to replace costly inversions of the point multiplication by additional field multiplications. Note that internally the computations for CURVE25519 only depend on the coordinates (x, z) and originally assumed $z = 1$. Hence, in order to randomize the point without affecting the final result, it has to be multiplied externally by a random 255-bit factor λ. More precisely, the value λ is randomly chosen 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. In addition, the core no longer can assume $z_{\mathcal{P}} = 1$ which mainly affects the computation of x_5 (see Eq. 7) where the final multiplication with z_3 cannot be skipped. Besides, all other values and constants of the computation are unaffected.

Memory Address Scrambling For our implementation, all inputs of the Montgomery ladder algorithm ($\mathcal{Q}_1$, $\mathcal{Q}_2$, and $\mathcal{Q}_3$) are stored in the same BRAM, and after each execution step, both results $\mathcal{Q}_4 = 2 \cdot \mathcal{Q}_1$ and $\mathcal{Q}_5 = \mathcal{Q}_1 + \mathcal{Q}_2$ will overwrite $\mathcal{Q}_1$ and $\mathcal{Q}_2$ while serving as a new input to the next iteration of the algorithm execution. For every step of the double-and-add computation, a single bit of the secret scalar is evaluated and indicates whether the inputs have to be swapped or not. Thus, the BRAM access pattern at the beginning of every execution of the double-and-add formula is key dependent and can help an attacker to reveal information of the secret scalar. Therefore, an adversary who is able to recover information on the BRAM addressing pattern using DPA can easily identify all inputs of the algorithm and thereby recover each bit of the secret scalar successively.

Hence, in order to protect the memory accesses against SCA and hide revealing access patterns, the addresses can be randomized for each execution. In the case of our design, we decided to choose random addresses for the intermediate values and all results that are stored in the memory prior to each point multiplication call. After every execution, we then choose a new set of random addresses for the intermediate values and results that are stored in the BRAMs. Since all addresses of our BRAM have a size of 6 bits, we can select up to 2^6 different sets of 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 base address. Therefore, we take 6 bits, supplied by an external RNG, and use them as seed (i.e., the base address) for the LFSR. Before our core can perform a new point multiplication operation, the LFSR is advanced and the random addresses are saved for the next point multiplication, initializing and preparing the BRAM by rearranging all required values and constants. Hence, 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, an attacker needs to distinguish between these memory lines.

Continuous Point Randomization The integration of all previously described countermeasures provides protection against SCA in particular if an adversary can observe multiple runs of the scalar multiplication operation. In a common adversarial setting, the attacker would observe multiple runs using the same scalar. In such a scenario, using randomized scalars, coordinates, and addresses mitigates the risk of disclosing sensitive and secret information. However, horizontal attacks pose another serious threat in particular for ECC [6]. In general, horizontal attacks — unlike previously described vertical attacks — exploit and combine the leakage within single traces in order to extract sensitive information. Unfortunately, scalar blinding does not provide protection against these kinds of threats, since the scalar is only randomized after each scalar multiplication operation [19]. However, re-randomizing the projective coordinate representation of either $\mathcal{Q}_4$ or $\mathcal{Q}_5$ after each iteration of the Montgomery ladder algorithm can help to mitigate the threat of horizontal attacks and increase the physical protection level. But certainly, the overall performance will be decreased due to two additional multiplication steps after each iteration in order to re-randomize the intermediate points and results. In addition, the continuous randomization yields in higher demands for fresh entropy and randomness for each scalar multiplication in general and each Montgomery ladder step in particular. To this end, we provide two different implementations, one using simple coordinate randomization that is only applied after each scalar multiplication while the second design uses continuous point randomization. In particular, we opted to re-randomize the intermediate point $\mathcal{Q}_5$ using two additional modular multiplications to compute $\lambda \cdot x_5$ and $\lambda \cdot z_5$ after each Montgomery ladder iteration with λ being a fresh random value for each re-randomization step.

Table 1 Resource utilization and performance results for CURVE25519 after PAR

	Aspect	Single core	
Utilization	*Component*	*Instances*	*Ratio*
	Look-up tables	2 077	3.90%
	Flip-flops	4 223	3.97%
	Logic slices	1 006	7.56%
	Digital signal processors	20	9.09%
	Block-RAMs	2	1.43%
Performance	*Operation*	*Cycles*	*Time (at 200 MHz)*
	Addition/subtraction	10	0.05 μs
	Multiplication	55	0.28 μs
	Montgomery step[1]	262	1.31 μs
	Montgomery step[2]	352	1.76 μs
	Inversion	14 372	71.86 μs
	Point multiplication[1]	114 980	574.90 μs
	Point multiplication[2]	149 540	747.70 μs

[1]Without point re-randomization, [2]With point re-randomization

3.3 Results

In this section, we present implementation and performance results after PAR for our CURVE25519 design. All results were obtained for a Xilinx XC7Z020CLG484-3 FPGA using the Vivado Design Suite 2015.4.

Table 1 provides implementation and utilization results of our presented side-channel protected, high-performance, single-core CURVE25519 architecture. Considering multi-core architecture with many cores operating in parallel, obviously, the limiting factor of this design is the number of DSP cores that is available for the given FPGA platform.

Besides, Table 1 also provides details on the performance of our proposed CURVE25519 architecture. Since the design can run at a maximum frequency of 200 MHz, a single iteration of the Montgomery ladder takes about $1.3\mu s$ (262 clock cycles), respectively, $1.8\mu s$ (352 clock cycles) including continuous point re-randomization as protection against horizontal attacks. Since our core performs 384 iterations for a single point multiplication, followed by a final inversion step, this leads to a final latency of about $575\mu s$, respectively, $748\mu s$ for a single operation and roughly 1 700 or 1 300 operations per second, clearly outperforming our initial goal of 1 000 operations per second for CURVE25519.

4 Side-Channel Protected Curve448

As mentioned before, CURVE25519 and CURVE448 share many similarities. However, in particular, the different structures of the underlying primes and the extended field size of CURVE448 forced us to completely rethink and revise the architectural options and decisions. Eventually, we ended up with an entirely new design that varies widely from the previous CURVE25519 architecture.

4.1 Design Rationales

Hardware implementations of modern cryptographic primitives can be designed in many ways and optimized from different perspectives, such as performance and throughput, resource utilization and hardware footprint, and physical and side-channel security. In this section, we discuss several optimization strategies and different approaches in order to achieve the chosen objectives for our final optimized implementation. In particular, we consider an efficient mapping to existing hardware structures before we optimize for high-performance applications and physical protection. To the best of our knowledge, this is one of the first hardware implementations of an elliptic curve with security level of more than 128 bits, so we can mainly compare to implementations targeting lower security requirements which are naturally faster. However, implementing high-security side-channel protected ECC must not prevent us from providing a competitive hardware architecture particularly if we can exploit the given structures optimally.

4.1.1 Field Arithmetic Optimizations

According to Section 2.1.2, all operations on elliptic curves over $GF(p)$ are based on operations over finite fields, such as modular addition, subtraction, multiplication, and inversion. Hence, the fundamental component of our hardware architecture is the Field Arithmetic Unit (FAU) (cf. Section 4.2) which has to be designed and optimized thoroughly before implementation since its operation is crucial and will impact to a great extent the final performance. Due to this fact, we had to do the entire optimization manually without relying on automated tools. In the following, we present and discuss several design choices leading to our high-performance FAU in particular optimized for CURVE448 on Xilinx FPGAs.

Modular Inversion In general, hardware designers have to decide whether performing modular inversion using dedicated units, e.g., based on the binary EEA, or re-using modular multiplication to compute the inversion using FLT or the ISR tweak. We decided against a dedicated unit due to its bad ratio between performance gain and area increase (for a single core); instead, we intend to perform modular inversion using the ISR tweak in Eq. 4.

Modular Multiplication With regard to the modular multiplication, we investigated and compared two different approaches: *Karatsuba Multiplication* and *Schoolbook Multiplication with Interleaved Reduction*.

Karatsuba. The Karatsuba algorithm, as shown in Eq. 3, uses decomposition in order to perform multiplications in partial steps which are more favorable due to smaller operand sizes. This approach allows to reduce the size of the multiplication unit (in terms of occupied area) but at cost of additional latency and control overhead.

Schoolbook. Due to the Solinas prime, the common schoolbook multiplication can be combined with an interleaved reduction step based on the fact that $2^{448} \equiv 2^{224} + 1 \pmod{p}$. Hence, each multiplication step is interleaved by a reduction before the partial products are accumulated.

Since the main goals of this work are efficient mapping, high-performance, and physical security for an elliptic curve *point multiplication*, we decided to build a modular multiplication unit based on the *Schoolbook Multiplication with Interleaved Reduction*. This approach has a lower latency compared to the *Karatsuba Multiplication* and more important, it perfectly maps to the available hardware resources (i.e., the DSP units) as shown in Section 4.2. Note, however, that both approaches need a final processing and reduction step which is handled by another, dedicated reduction unit.

Although modern FPGAs provide asymmetric multiplication cores that allow to build more efficient multipliers [20], we instantiated only 16×16 multipliers for the sake of symmetry and easy combination of the multiplication with interleaved reduction.

4.1.2 Architectural Customizations

In Section 2.5, we provide a general set of optimization strategies that we considered throughout the entire design process. Based upon this, we decided to extend the internal data path of our architecture to a full width of 448 bits in order to allow maximum parallelization and optimal performance within all sub-modules and to avoid stalls during operation. In general, this allows to provide the FAU with all operands on time avoiding or reducing idling phases due to load or store operations.

Further, since most of the time, our FAU will perform modular multiplications and each multiplication is performed in 28 steps, thus requiring at least 28 cycles, we implement the optimal number of additional pipeline stages within the DSP units (i.e., four) in order to allow operation at maximum frequency. Note, however, that this modification as well increases latency of modular additions and subtractions, since we share and re-use the DSP units for these operations.

Eventually, our FAU is divided into sub-modules which are separated by register stages in order to allow independent operation and in order to decrease the critical path.

4.2 Implementation

In this section, we provide implementation details of our CURVE448 hardware architecture, in particular the arithmetic units of our enhanced design which provides further protection mechanism and countermeasures against side-channel attacks. Most of all, we like to point out that this additional protection can be integrated smoothly with only minimum effort and resource overhead.

4.2.1 Field Arithmetic Unit

As mentioned earlier, the FAU (as sketched in Fig. 3) is the basic component of our architecture. In principle, it can be subdivided into two different components, an arithmetic core (AC) and a reduction core (RC), that we explain in detail in the following.

Arithmetic Core (AC). In order to perform arithmetic operations, the body of our AC is implemented using 28 fully pipelined DSPs in parallel. Each DSP provides different configurations, such as 16-bit addition, 16-bit subtraction, or 16×16-bit multiplication. By selecting the according configuration during run time, the AC implements full data path additions, subtractions, and multiplications. Hence, sharing the DSPs allows implementing *Carry-Save Additions* and *Subtractions* as well as *Schoolbook Multiplications with Interleaved Reduction* within a single unit. The final accumulation and reduction of the results is performed by the RC.

Reduction Core (RC). The RC consists of another five DSPs, three to pre-add partial products and two to accumulate the carries and to perform the final reduction. In general, the reduction process is performed serially on

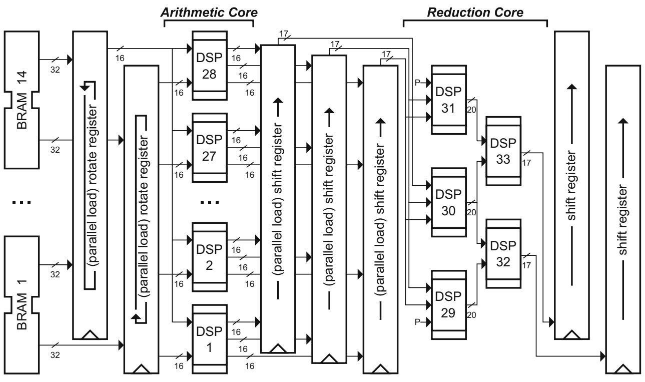


Fig. 3 Simplified architecture of the FAU

17-bit words in order to take advantage of the internal shift operation of the DSPs. However, due to the serial design, the reduction is the bottleneck of the FAU. Still, both, the AC and the RC, can be operated independently and in parallel in order to alleviate the disadvantage of the serial design.

4.2.2 Side-Channel Countermeasures

In this section, we explain the necessary modifications of our architecture in order to provide a basic protection against side-channel attacks using *point randomization* and *scalar blinding*.

Scalar Blinding According to Section 2.4.2, the scalar is blinded by a random multiple of the group order and due to the special structure of the underlying prime, it is advisable to use random factors of at least half of the field size [25]. Hence, the secret scalar is blinded and expanded by 224 bits and the core itself accepts scalars of up to 672 bits. However, based on the increased scalar size, the final performance of the implementation naturally drops due to 224 additional Montgomery steps that have to be performed compared to an unprotected architecture.

Simple Point Randomization As explained in Section 2.4.1, *point randomization* uses a randomly chosen parameter λ which is applied to the base point. In general, this process initializes the z-coordinate with λ, but the x-coordinate has to be multiplied by λ externally. Hence, this countermeasure can be integrated easily, since the core already accepts base points in projective coordinate representation.

Continuous Point Randomization As already mentioned in Section 3.2.3, in particular the continuous re-randomization of intermediate points based on our point randomization countermeasure can mitigate the risk of horizontal attacks against our implementation. To this end, including two additional modular multiplications at the end of each Montgomery ladder iteration in order to randomize the intermediate point Q_5 can help to prevent successful leakage exploitation using horizontal attacks. Of course, the additional security and protection mostly comes at cost of increased latency and additional requirement for fresh randomness during every iteration.

4.3 Results

In this section, we present implementation and performance results after PAR for our CURVE448 design. Note, that all results were obtained for a Xilinx XC7Z020CLG484-3 FPGA using the Vivado Design Suite 2015.4.

Table 2 Resource utilization and performance results for CURVE448 after PAR

	Aspect	Single core	
Utilization	*Component*	*Instances*	*Ratio*
	Look-Up tables	4 624	8.69%
	Flip-flops	8 209	7.72%
	Logic slices	1 985	14.92%
	Digital signal processors	33	15.00%
	Block-RAMs	14	10.00%
Performance	*Operation*	*Cycles*	*Time (at 341 MHz)*
	Addition/subtraction	4	11.72 ns
	Multiplication	32	93.76 ns
	Reduction	36	105.48 ns
	Montgomery step[1]	694	2.03 μs
	Montgomery step[2]	766	2.25 μs
	Inversion	32 976	96.62 μs
	Point multiplication[1]	499 344	1.46 ms
	Point multiplication[2]	547 728	1.61 ms

[1]Without point re-randomization, [2]With point re-randomization

Table 2 reports the final implementation and performance results after PAR for our protected, high-performance, single-core CURVE448 architecture. Similar to the CURVE25519 architecture, again the number of available DSP resources is the limiting factor for multi-core designs.

Besides, particularly, the maximum frequency of 341 MHz is noteworthy, which allows to perform a single point multiplication, including 672 iterations of the Montgomery ladder and a final inversion, within 1.46ms respectively 1.61ms (including protection against horizontal attacks) despite the large field size of 448 bits. Eventually, this leads to a final performance of 683 or 622 operations per second, still outperforming our initial goal of 500 point multiplications per second.

5 Comparison and Discussion

To the best of our knowledge, there are only few implementations of ECC schemes for FPGA devices which target a security level equal or above 128 bits that are publicly available. Along with the fact that modern FPGA architectures have evolved a lot and now include more powerful features compared to earlier generations, a fair and meaningful discussion or comparison of different designs and implementations with previous work is not straightforward. Nevertheless, we like to put our results in the context with existing implementations to allow the reader a quick overview on other designs and architectures.

Table 3 lists different FPGA architectures of recent ECC schemes over prime fields, including our previous architectures of CURVE25519 [22, 23] and CURVE448 [24], NIST P-256 [9], and recent results on FOUR$\mathbb{Q}$ [11]. Note, however, that all of these implementations target a lower security level of about 128 bits, except for CURVE448. This particularly implies a smaller field size and thus less arithmetic complexity for the basic operations.

Although our architectures are based on the results of [23] and [24], area and performance numbers are to some extent different. In particular, for CURVE25519, we could reduce the number of logic slices (LSs) significantly, simply by using the Vivado Design Suite for PAR instead of ISE 14.7. On the other hand, the latency, i.e., the number of clock cycles per operation, increased significantly due to the extended size of the secret scalar (384 bits instead of 269 bits). In addition, since we had to include an additional multiplication by z_3, this increased the latency as well (for both, CURVE25519 and CURVE448). Besides, since the randomization of scalar and point is provided externally, we could reduce the number of DSPs for both designs.

6 Side-Channel Evaluation

Eventually, we performed a practical, test-based side-channel Analysis and leakage assessment in order to evaluate our implementations and included countermeasures. To this end, we performed practical power measurements on a SAKURA-X [14] side-channel evaluation board. Our designs were running on a 7-series Kintex FPGA (XC7K160T) using a stable, jitter-free 96 -MHz clock signal. More precisely,, we measured the voltage drop over an 1 Ω resistor in the V_{dd} path by means of a digital oscilloscope using a sampling rate of 250 MS/s and 20 MHz bandwidth limitation.

6.1 Detection of Scalar-Dependent Leakage

During our first analysis, we first focused on detection of scalar-dependent leakage using a fixed base point throughout the evaluation process while the scalar is chosen from a set of random scalars. In order to apply a non-specific t test, the measurements are randomly interleaved with operations

Table 3 Comparison of different designs for elliptic curve cryptography over prime fields on FPGAs

Scheme	Device	Security	Architecture				Performance			Ref.
			Data path	LS	DSP	BRAM	Cycles	MHz	OP/s	
NIST P-256										
Single-core	XCC4VFX12	128 bits	32 bits	1715	32	11	243 000	490	2 020	[9]
Multi-core	XCC4VFX12	128 bits	32 bits	24 574	512	176	–	375	24 700	[9]
FOUR$\mathbb{Q}$										
Single-core[4]	XC7Z020	123 bits	127 bits	565	16	7	58 967	190	3 222	[11]
Single-core[5]	XC7Z020	123 bits	127 bits	1 691	27	10	29 739	190	6 389	[11]
Multi-core[5]	XC7Z020	123 bits	127 bits	5 697	187	110	–	175	64 730	[11]
CURVE25519										
Single-core[1,4]	XC7Z020	127 bits	34 bits	1 029	20	2	79 400	200	2 519	[22]
Single-core[2,4]	XC7Z020	127 bits	34 bits	1 169	22	2	83 252	200	2 402	[23]
Single-core[2,4]	XC7Z020	127 bits	34 bits	1 006	20	2	114 980	200	1 739	This work
Single-core[3,4]	XC7Z020	127 bits	34 bits	1 176	20	2	149 540	200	1 337	This work
CURVE448										
Single-core[1,4]	XC7Z020	224 bits	448 bits	1 580	33	14	328 286	357	1 087	[24]
Single-core[2,4]	XC7Z020	224 bits	448 bits	1 648	35	14	473 926	335	708	[24]
Single-core[2,4]	XC7Z020	224 bits	448 bits	1 985	33	14	499 344	341	683	This work
Single-core[3,4]	XC7Z020	224 bits	448 bits	2056	33	14	547 728	341	622	This work

[1]Unprotected, [2]Scalar blinding and simple point randomization, [3]Scalar blinding and continuous point randomization, [4]Montgomery ladder, [5]Endomorphism

based on a fix scalar. Further, in order to avoid any input-dependent leakage, the scalar blinding and point randomization have been performed externally while the cores only accept masked inputs, i.e., blinded scalars and points in projective coordinate representation. However, the final result of the scalar-point multiplication still is available only in an unprotected state which certainly results in output-dependent leakage. Hence, to avoid such result-depending detection of leakage, we exclude the final operations after the inversion from our measurements and observations.

Further on, we defined four different evaluation profiles in order to gradually evaluate our main countermeasures (i.e., *point randomization* and *scalar blinding*). In particular, Profile 1 provides a reference since all countermeasures are disabled, i.e., the PRNG does not provide random values but all zero. Profile 2 and 3 then investigate the *scalar blinding* and *point randomization* countermeasures individually before Profile 4 evaluates our fully protected designs with both countermeasure active in combination.

During our analysis of CURVE25519 and CURVE448 , we used the default base points $x_{\mathcal{P}} = 9$ and $x_{\mathcal{P}} = 5$, and performed a non-specific t test with fix and randomly chosen scalars according to [26]. In Fig. 4, we provide two mean traces for CURVE25519, respectively, CURVE448, both based on 1000 measurements, where we at first disabled (Fig. 4a, c) and then enabled (Fig. 4b, d) all of our countermeasures.

6.1.1 Profile 1: PRNG Off

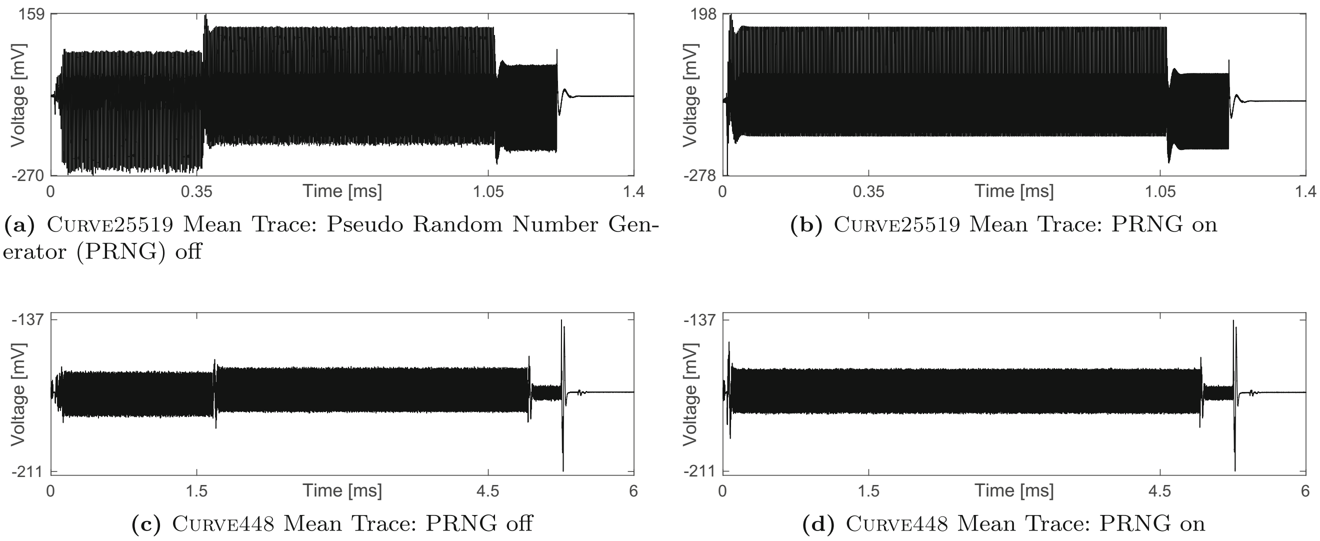
Our first profile serves as a reference setup where we disabled our PRNG in order to provide a constant zero value. Consequently, the secret scalar is provided in an unblinded fashion and the base point has a constant z-coordinate of $z_P = 1$. In Figs. 5a and 6a, we present the t test-based evaluation results after measuring 10 000 power traces. Obviously, the t test detects scalar-dependent leakage with very high confidence for both architectures giving proof for the correctness of our setup. As expected, the leakage is mainly detectable during the last 255, respectively, 448 Montgomery step executions and the final inversion, since the secret scalar is padded with zeros for both cases (fix and random scalar).

6.1.2 Profile 2: Point Randomization

The second evaluation profile investigates the individual security gain through the randomization of projective coordinate representations without scalar blinding countermeasures. Therefore, the core is provided with a random multiple of the $x'_{\mathcal{P}} = 9\lambda \pmod{p}$ or $x'_{\mathcal{P}} = 5\lambda \pmod{p}$ and the random z-coordinate $z' = \lambda$. For this profile, we captured 100 000 power traces with fix vs. random scalars and the t test results are shown in Figs. 5b and 6b. For both measurements, since the unprotected scalar is expanded by 128 bits, respectively, 224 bits, and padded with leading zeros, we can clearly distinguish an initial phase that processes these zeros from the actual processing of the scalar bits due to its smaller t values. In summary, randomizing projective coordinate representations helps to decrease the observable side-channel leakage though it does not prevent the implementation from exhibiting scalar-dependent side-channel information.

6.1.3 Profile 3: Scalar Blinding

In Figs. 5c and 6c, we provide the evaluation results for our third profile in order to investigate our scalar blinding countermeasure. Again, we measured 100 000 power traces



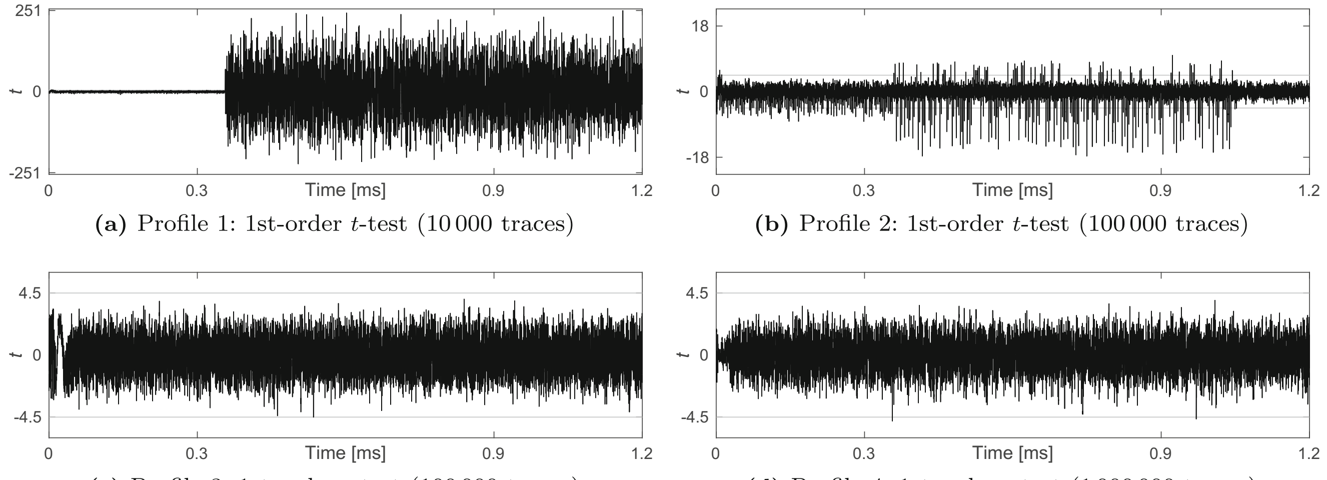
(a) CURVE25519 Mean Trace: Pseudo Random Number Generator (PRNG) off

(b) CURVE25519 Mean Trace: PRNG on

(c) CURVE448 Mean Trace: PRNG off

(d) CURVE448 Mean Trace: PRNG on

Fig. 4 Mean traces over 1 000 power traces



(a) Profile 1: 1st-order t-test (10 000 traces) **(b)** Profile 2: 1st-order t-test (100 000 traces)

(c) Profile 3: 1st-order t-test (100 000 traces) **(d)** Profile 4: 1st-order t-test (1 000 000 traces)

Fig. 5 Non-specific t test results for CURVE25519: fix vs. random scalar

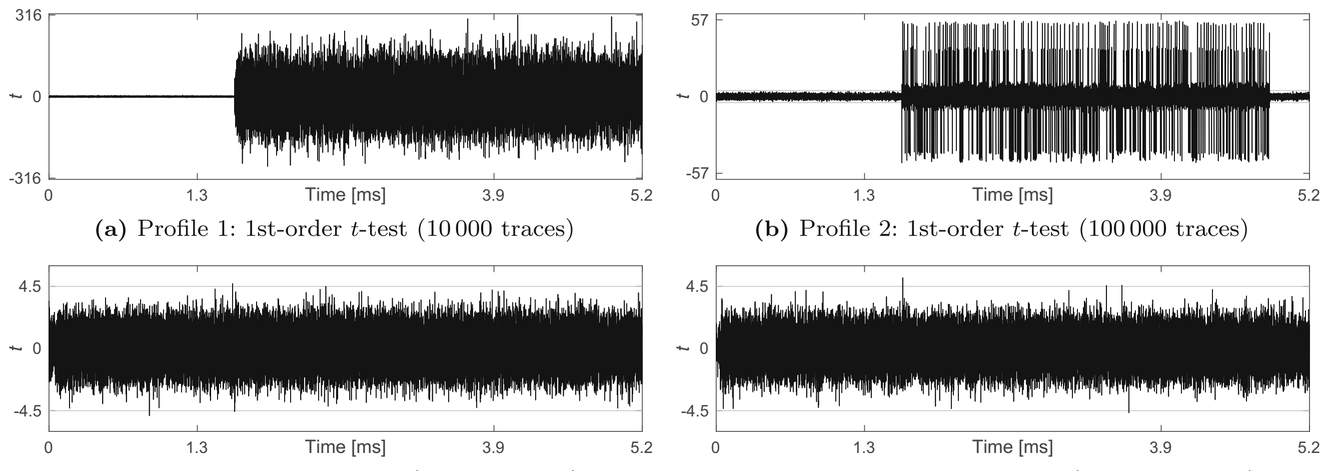
using a non-specific t test setup with fix vs. random scalars. In addition, each scalar is blinded externally using a random 128-bit or 224-bit blinding factor (i.e., about half of the bit-length of the according group order) which is multiplied by the group order of the base point and added to the original scalar. Then, the core receives and processes only the blinded scalar. Obviously, this countermeasure avoids any detectable scalar-dependent leakage both designs and for the given number of measurements and in particular we cannot distinguish the loading and processing of any fix and random secret scalar (on the first statistical order).

6.1.4 Profile 4: Combination

Eventually, our forth profile combines all countermeasures in order to investigate their interaction. Figures 5d and 6d provide the non-specific t test results using 1 000 000 power traces and as expected we cannot observe any first-order side-channel leakage for both designs while using scalar blinding and randomized projective coordinates.

6.2 Detection of Base-Point-Dependent Leakage

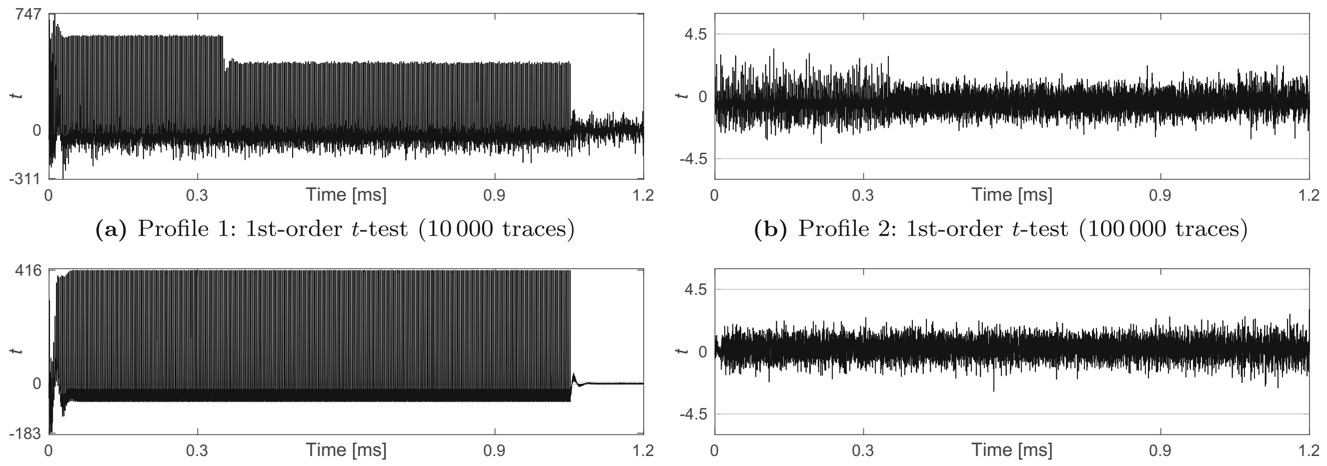
In a further step, we evaluated the detection of base-point-dependent leakage, this time using a fixed scalar throughout the evaluation procedure while the base point is chosen from a random set and interleaved with operations on a fix base point. Again, we used $x_{\mathcal{P}} = 9$ and $x_{\mathcal{P}} = 5$ as fix base points. However, in order to ensure the correct operation of our core and the scalar blinding countermeasure, the random base point has to have the same group order than the default base point. To this end, we modified our measurement setup



(a) Profile 1: 1st-order t-test (10 000 traces) **(b)** Profile 2: 1st-order t-test (100 000 traces)

(c) Profile 3: 1st-order t-test (100 000 traces) **(d)** Profile 4: 1st-order t-test (1 000 000 traces)

Fig. 6 Non-specific t test results for CURVE448: fix vs. random scalar



(a) Profile 1: 1st-order t-test (10 000 traces) (b) Profile 2: 1st-order t-test (100 000 traces) (c) Profile 3: 1st-order t-test (10 000 traces) (d) Profile 4: 1st-order t-test (1 000 000 traces)

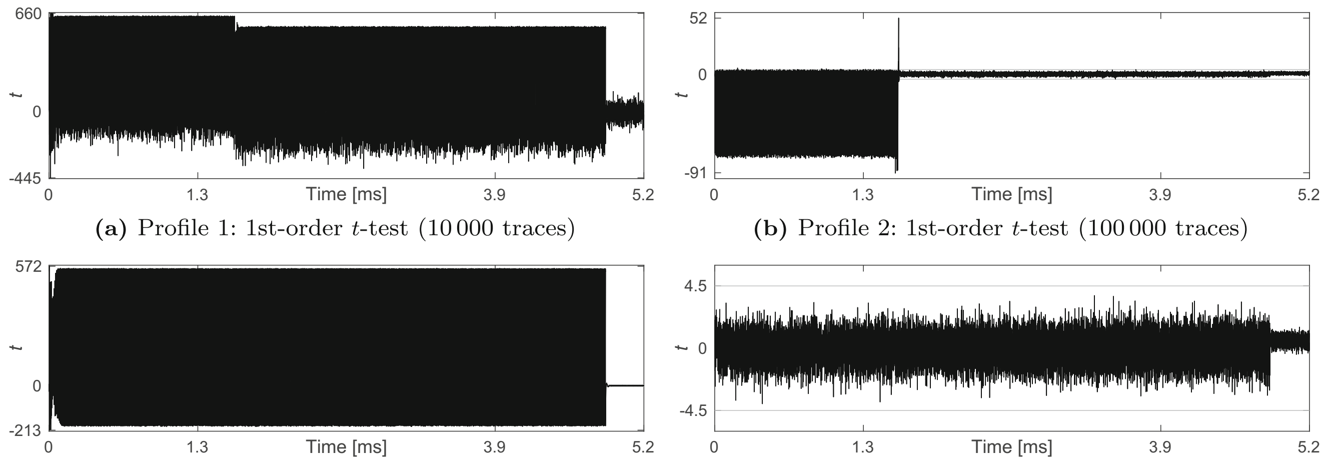
Fig. 7 Non-specific t test results for CURVE25519: fix vs. random base point

such that when a random base point is sent to the core, an arbitrarily chosen scalar s is used to derive a new base point as $\mathcal{P}' = s \times \mathcal{P}$, i.e., we performed an additional point multiplication to ensure the same group order for $\mathcal{P}$ and $\mathcal{P}'$. Besides, the scalars have been fixed to the test vectors that are given in RFC 7748.

Profile 1: PRNG Off Again, the first profile is intended as a reference for our evaluation setup. To this end, we again disabled our PRNG and countermeasures. Consequently, the scalar is not blinded and the base point has a constant z-coordinate of $z_{\mathcal{P}} = z_{\mathcal{P}'} = 1$. Figures 7a and 8a present the t test results using 10 000 power traces. As expected, the t score exceeds the threshold significantly, i.e., we can detect base-point-dependent leakage with high confidence.

Profile 2: Point Randomization In Figs. 7b and 8b, we provide the evaluation results when enabling point randomization as countermeasure. Obviously, we cannot detect any first-order side-channel leakage for CURVE25519 after measuring 100 000 power traces, but surprisingly we detect first-order leakage for CURVE448. However, the detected leakage only appears during the first 224 iterations of the Montgomery steps, i.e., where the secret scalar is padded with leading zeros. Hence, without the padding or with using a randomized representation of the scalar , we expect to prevent this detection of leakage.

Profile 3: Scalar Blinding Figures 7c and 8c provide the evaluation results of our third profile, when enabling the scalar blinding countermeasure, using only 10 000 power traces and with fix scalar. In contrast to Profile 3 with



(a) Profile 1: 1st-order t-test (10 000 traces) (b) Profile 2: 1st-order t-test (100 000 traces) (c) Profile 3: 1st-order t-test (10 000 traces) (d) Profile 4: 1st-order t-test (1 000 000 traces)

Fig. 8 Non-specific t test results for CURVE448: fix vs. random base point

fix base point, we can clearly detect and observe leakage for both architectures. In particular, the internal processing of the point $Q_3 = Q_1 - Q_2$ can be detected due to its dependency on the base point since $x_3 = x_{\mathcal{P}}$ and $z_3 = z_{\mathcal{P}}$ hold for each iteration of the Montgomery ladder. Hence, we did not expect to prevent base-point-dependent leakage when blinding the secret scalar.

Profile 4: Combination The last profile again combines both countermeasures in order to analyze and evaluate the interaction of both mechanisms. In Figs. 7d and 8d, we present the evaluation results, again using 1 000 000 power traces. As we expect, we cannot observe first-order side-channel leakage for our CURVE25519 and CURVE448 architectures while both countermeasures are active.

7 Conclusion

In this work, we presented FPGA architectures for CURVE25519 and CURVE448 that provide high-performance ECC operations (point multiplications) including advanced protection against physical attacks. In particular, we have proven that both elliptic curves can be efficiently mapped to existing FPGA structures although only software implementations were considered during the design phase of both curves. In addition, our architectures support virtually all high-performance applications for all security levels in the range of 128–224 bits, providing more than 1 700 and 600 operations per second for CURVE25519 and CURVE448 on a mid-range Xilinx XC7Z020 FPGA. Eventually, our leakage assessment confirms with high confidence, that even with 1 000 000 power measurements we cannot detect any scalar- or base-point-dependent leakage while the *scalar blinding* and *point randomization* countermeasures are active.

References

1. Agrawal D, Archambeault B, Rao JR, Rohatgi P (2002) The EM side-channel(s). In: 4th International workshop on cryptographic hardware and embedded systems - CHES 2002. Redwood Shores, CA, USA, Revised Papers, pp 29–45
2. Alrimeih H, Rakhmatov DN (2014) Fast and flexible hardware support for ECC over multiple standard prime fields. IEEE Trans VLSI Syst 22(12):2661–2674
3. Bernstein DJ (2006) Curve25519: new Diffie-Hellman speed records. In: 9th International Conference on theory and practice of public-key cryptography on public key cryptography - PKC 2006. New York, NY, USA, April 24-26, 2006, proceedings, volume 3958 of lecture notes in computer science. Springer, pp 207–228
4. Coron J-S (1999) Resistance against differential power analysis for elliptic curve cryptosystems. In: 1st International workshop on cryptographic hardware and embedded systems - CHES 1999. Worcester, MA, USA, August 12-13, 1999, proceedings, volume 1717 of lecture notes in computer science. Springer, pp 292–302
5. de Dormale GM, Quisquater J-J (2007) High-speed hardware implementations of elliptic curve cryptography: a survey. Journal of Systems Architecture
6. Dugardin M, Papachristodoulou L, Najm Z, Batina L, Danger J-L, Guilley S (2016) Dismantling real-world ECC with horizontal and vertical template attacks. In: Constructive side-channel analysis and secure design - 7th international workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, revised selected papers, volume 9689 of lecture notes in computer science. Springer, pp 88–108
7. Fan J, Xu G, De Mulder E, Schaumont P, Preneel B, Verbauwhede I (2010) State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures. In: IEEE International symposium on hardware oriented security and trust - HOST 2010, Anaheim Convention Center, CA, USA, June 13-14, 2010, proceedings. IEEE Computer Society, pp 76–87
8. Fan J, Verbauwhede I (2012) An updated survey on secure ECC implementations attacks, countermeasures and cost. In: Cryptography and security: from theory to applications - essays dedicated to Jean-Jacques Quisquater on the occasion of his 65th birthday, volume 6805 of lecture notes in computer science. Springer, pp 265–282
9. Güneysu T, Paar C (2008) Ultra high performance ECC over NIST primes on commercial FPGAs. In: 10th International workshop on cryptographic hardware and embedded systems - CHES 2008. Washington, D.C., USA, August 10-13, 2008, proceedings, volume 5154 of lecture notes in computer science. Springer, pp 62–78
10. Hamburg M (2015) Ed448-Goldilocks, a new elliptic curve. IACR Cryptology ePrint Archive, 2015:625. http://eprint.iacr.org/2015/625
11. Järvinen K, Miele A, Azarderakhsh R, Patrick L (2016) Four$\mathbb{Q}$ on FPGA: new hardware speed records for elliptic curve cryptography over large prime characteristic fields. In: 18th International conference on cryptographic hardware and embedded systems - CHES 2016. Santa Barbara, CA, USA, August 17-19, 2016, proceedings, volume 9813 of lecture notes in computer science. Springer, pp 517–537
12. Kocher PC (1996) Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: 16th Annual international cryptology conference on advances in cryptology - CRYPTO '96. Santa Barbara, California, USA, proceedings, pp 104–113
13. Kocher PC, Jaffe J, Jun B (1999) Differential power analysis. In: 19th Annual international cryptology conference on advances in cryptology - CRYPTO '99. Santa Barbara, California, USA, Proceedings, pp 388–397
14. UEC Satoh Lab. Side-channel attack user reference architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html
15. Montgomery PL (1987) Speeding the Pollard and elliptic curve methods of factorization. Math Comput 48(177):243–264
16. De Mulder Elke, Örs SB, Preneel B, Verbauwhede I (2007) Differential power and electromagnetic attacks on a FPGA implementation of elliptic curve cryptosystems. Comput Electric Eng 33(5–6):367–382
17. Orlando G, Paar C (2001) A scalable GF(p) elliptic curve processor architecture for programmable hardware. In: 3rd International workshop on cryptographic hardware and embedded systems - CHES 2001. Paris, France, May 14-16, 2001, Proceedings, volume 2162 of lecture notes in computer science. Springer, pp 348–363
18. Örs SB, Batina L, Preneel B, Vandewalle J (2003) Hardware implementation of an elliptic curve processor over GF(p). In: 14th IEEE International conference on application-specific systems, architectures, and processors - ASAP 2003. The Hague, The

Netherlands, June 24-26, 2003, Proceedings. IEEE Computer Society, pp 433–443

19. Poussier R, Zhou Y, Standaert F-X (2017) A systematic approach to the side-channel analysis of ECC implementations with worst-case horizontal attacks. In: 19th International conference on cryptographic hardware and embedded systems - CHES 2017. Taipei, Taiwan, September 25-28, 2017, Proceedings, volume 10529 of lecture notes in computer science. Springer, pp 534–554
20. Roy DB, Mukhopadhyay D, Izumi M, Takahashi J (2014) Tile before multiplication: an efficient strategy to optimize DSP multiplier for accelerating prime field ECC for NIST curves. In: The 51st Annual design automation conference 2014, DAC '14. San Francisco, CA, USA, June 1-5, 2014, pp 177:1–177:6
21. Sakiyama K, Mentens N, Batina L, Preneel B, Verbauwhede I (2006) Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems. In: 2nd International Symposium on reconfigurable computing: architectures, tools and applications - ARC 2006. Delft, The Netherlands, March 1-3, 2006, proceedings, volume 3985 of lecture notes in computer science. Springer, pp 347–357
22. Sasdrich P, Güneysu T (2014) Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices. In: 10th International Symposium on reconfigurable computing: architectures, tools and applications - ARC 2014. Vilamoura, Portugal, April 14-16, 2014, proceedings, volume 8405 of lecture notes in computer science. Springer, pp 25–36
23. Sasdrich P, Güneysu T (2015) Implementing Curve25519 for side-channel-protected elliptic curve cryptography. ACM Trans Reconfig Technol Syst - TRETS 9(1):3
24. Sasdrich P, Güneysu T (2017) Cryptography for next generation TLS - implementing the RFC 7748 elliptic Curve448 cryptosystem in hardware. In: Proceedings of the 54th design automation conference - DAC 2017. Austin, TX, USA, June 18-22, 2017. ACM, pp 1–6
25. Schindler W, Wiemers A (2015) Efficient side-channel attacks on scalar blinding on elliptic curves with special structure. In: NIST Workshop on ECC standards
26. Tunstall M, Goodwill G (2016) Applying TVLA to public key cryptographic algorithms. IACR Cryptology ePrint Archive, 2016:513. http://eprint.iacr.org/2016/513