Cryptography for Next Generation TLS:

Implementing the RFC 7748 Elliptic Curve448 Cryptosystem in Hardware

Pascal Sasdrich
Horst Görtz Institute for IT Security,
Ruhr-Universität Bochum
pascal.sasdrich@rub.de

Tim Güneysu
University of Bremen & DFKI
tim.gueneysu@uni-bremen.de

ABSTRACT

With RFC 7748 the two elliptic curves Curve25519 and Curve448 were proposed for the next generation of TLS. Both curves were designed and optimized purely for software implementation; their implementation in hardware or physical protection against side-channel attacks were not considered in the design phase. Recently, it has been shown that for Curve25519 an efficient implementations in hardware along with side-channel protection is feasible – yet results for the high-security Curve448 are missing. In this work we demonstrate that Curve448 can indeed be efficiently and securely implemented in hardware. We present a novel architecture for Curve448 that can compute more than 1000 point multiplications per second with 1580 logic slices and 33 DSP units of a Xilinx XC7Z020 FPGA.

Keywords

Curve448, RFC 7748, FPGA, high-performance, implementation, side-channel, countermeasure

1. INTRODUCTION

Elliptic Curve Cryptography (ECC) has become the most predominant scheme in the area of asymmetric cryptography during the last decade. However, recent revelations on manipulations and backdoors 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, e.g., self-driving cars, high-performance TLS (including Curve25519 and Curve448) has to be implemented on various physical and embedded devices. But since both curves were primarily designed for software platforms (x86/x64) and without explicit evaluation of their resistance against low-level physical such as side-channel attacks, we still need to investigate the hardware implementation of both curves in combination with suitable countermeasures.

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 the author(s) 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.

DAC '17, June 18 - 22, 2017, Austin, TX, USA

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4927-7/17/06…$15.00

DOI: http://dx.doi.org/10.1145/3061639.3062222

Contribution. In this work, we present an efficient hardware implementation of Curve448 for modern Xilinx Field-Programmable Gate Array (FPGA) devices. Our work fills the gap left from the previous work on Curve25519 [14, 15]. Despite of the similarity of Curve448 and Curve25519, 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 on a mid-range Xilinx XC7Z020 FPGA, providing a performance of more than 1000 *point multiplications* per second at moderate resource costs.

Outline. This work is organized as follows: In Section 2 we summarize and discuss previous but related work of relevance before we give a brief introduction into the topic of ECC in Section 3, focusing on Curve448 in particular. Before presenting the final implementation details in Section 5, we discuss optimization strategies and justify our final design choices in Section 4. Eventually, implementation and performance results are summarized and presented in Section 6 before we conclude in Section 7.

2. PREVIOUS WORK

ECC has been proposed independently by Neal Koblitz [7] and Victor Miller [10] in 1985. Since then, it has established as a viable field of research in public-key cryptography and has been implemented in a great number of industrial products. Up to date, there exists a plethora of publications on hardware implementations that we will not cover or discuss within the scope of this work. Instead we restrict ourselves to briefly list the most relevant ones. However, for a detailed history and list of references we would like to refer the interested reader to [2].

As one of the first ECC architectures implemented on a FPGA, in 2001, Orlando and Paar [12] presented a design using pre-computations and Montgomery multiplications for improved performance results. Their work was followed by many more, including designs based on dedicated multi-

pliers [13] and integrated Digital Signal Processor (DSP) units embedded into modern FPGAs [4]. In 2014, Sasdrich and Güneysu [14, 15] presented an FPGA architecture for Curve25519, the first candidate of RFC 7748. Recently, Järvinen et al. [6] presented a high-performance architecture for Four$\mathbb{Q}$, a novel elliptic curve with about 128 bit security that supports highly-efficient point multiplications.

3. BACKGROUND

In this section we will briefly cover basic aspects of ECC. Note, however, that we will only focus on ECC over prime fields. Further, we give a short introduction to Curve448 and its specification.

3.1 Elliptic Curve Cryptography

Given an elliptic curve $\mathcal{E}$ over the Galois Field $GF(p)$ (with $p > 3$), defined by the Edwards equation

$$\mathcal{E}_d : y^2 + x^2 \equiv 1 + dx^2y^2 \pmod{p}, \tag{1}$$

we can define tuples of affine coordinates (x_i, y_i) as points $\mathcal{P}_i \in \mathcal{E}$ on the elliptic curve. Considering two points $\mathcal{P}_i, \mathcal{P}_j$, we can define the basic group operations as *point addition* if $i \neq j$ and *point doubling* if $i = j$. Moving from affine coordinate to projective coordinate representations, hence defining points on $\mathcal{E}$ as tuples (x_i, y_i, z_i), curve equations can be relaxed in order to reduce the number of costly operations (i.e., modular inversions).

Applying elliptic curves for cryptographic purposes relies on the Elliptic Curve Discrete Logarithm Problem (ECDLP) which states that, given two points $\mathcal{P}, \mathcal{Q} \in \mathcal{E}$, it is hard to find a scalar k such that $\mathcal{Q} = k \cdot \mathcal{P}$ holds, whereas the *point multiplication* is defined as a k-fold *point addition* of $\mathcal{P}$.

Eventually, the ECDLP is the fundamental problem that allows building cryptographic protocols and schemes based on elliptic curves. Common examples include the Elliptic Curve Diffie-Hellman (ECDH) key exchange, the Elliptic Curve Digital Signature Algorithm (ECDSA) and the ElGamal encryption scheme.

3.2 Specification of Curve448

Ed448-Goldilocks, in RFC 7748 denoted as Curve448, was proposed and designed by Mike Hamburg [5] in order to provide a high-security alternative (i.e., a field size between 384 and 521 bits) to existing NIST [9] and Brainpool curves [8]. The design process of Curve448 strictly heeded the SafeCurves policies and criteria [1] which should enable simple but secure implementations of an elliptic curve that meets all requirements.

3.2.1 Field Arithmetic

Curve448[1] is an untwisted Edwards curve defined by Equation 1, with a security level of 224 bits and factor $d = -39\,081$, the Solinas prime $p = 2^{448} - 2^{224} - 1$ obtained from its golden ratio $\phi = 2^{224}$. 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-bit) due to the fact that $448 = 56 \times 8 = 28 \times 16 = 14 \times 32 = 7 \times 64$.

[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.

Due to its golden ratio ϕ the Solinas prime 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:

$$C = A \cdot B = (a_0 + a_1\phi) \cdot (b_0 + b_1\phi) = \\ (a_0b_0 + a_1b_1) + ((a_0 + a_1) \cdot (b_0 + b_1) - a_0b_0)\phi \tag{2}$$

Eventually, modular inversion can be tweaked compared to modular inversions based on Fermat's Little Theorem (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:

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

3.2.2 Group Arithmetic

According to RFC 7748, the function `X448` performs a variable scalar-point multiplication on the Montgomery curve (Curve448) using the Montgomery ladder algorithm [11] that enables the computation of a *point addition* and a *point doubling* in a single combined step. Hence, given two points $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathcal{E}$ and their 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 *point doubling* with:

$$x_4 = (x_1 - z_1)^2 \cdot (x_1 + z_1)^2 \tag{4}$$

$$z_4 = 4x_1z_1 \cdot (x_1^2 + dx_1z_1 + z_1^2) \tag{5}$$

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

$$x_5 = z_3((x_1 - z_1) \cdot (x_2 + z_2) + (x_1 + z_1) \cdot (x_2 - z_2))^2 \tag{6}$$

$$z_5 = x_3((x_1 - z_1) \cdot (x_2 + z_2) - (x_1 + z_1) \cdot (x_2 - z_2))^2 \tag{7}$$

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. In total, a single step of the Montgomery ladder algorithm involves 6 multiplications, 4 squarings, 4 additions, and 4 subtractions over $GF(p)$.

3.2.3 Point Multiplication

Given a public point $\mathcal{P} \in \mathcal{E}$ and a secret 448-bit scalar 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 bitwise (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 448 steps.

3.3 Side-Channel Protection

Side-Channel Analysis (SCA) is a common attack to exploit information leakage related to internals of physical devices e.g., by inspecting and analyzing power consumption, electromagnetic radiations, or timing behavior of the device, in order to extract secret information. Hence, 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 inherently provides

basic protection against timing and Simple Power Analysis (SPA). However, dedicated countermeasures against more sophisticated attacks such as Differential Power Analysis (DPA) still need to be addressed. Since there is a wealth of different countermeasures against DPA attacks, we refer the interested reader to an overview of Fan et al. [3], but only introduce *point randomization* and *scalar blinding* as an example[2].

3.3.1 Point Randomization

SCA adversaries have physical access to cryptographic implementations and can at least observe inputs and outputs to the device including additional side-channel information. In order to prevent any extraction of cryptographic secrets using statistical analysis of the side-channel information, common countermeasures try to randomize the process data and the side-channel leakage. *Point randomization* 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 Equation 8, which yields in different point representations (depending on λ).

$$\mathcal{P}_r = (x_{\mathcal{P}_r}, z_{\mathcal{P}_r}) = (\lambda x_{\mathcal{P}}, \lambda z_{\mathcal{P}}) = (\lambda x_{\mathcal{P}}, \lambda) \tag{8}$$

Fortunately, *point randomization* does not affect the correctness of the underlying scheme, as proven in Equation 9.

$$x_{\mathcal{P}_r} \cdot z_{\mathcal{P}_r}^{-1} = \lambda x_{\mathcal{P}} \cdot \lambda^{-1} = x_{\mathcal{P}} \pmod{p} \tag{9}$$

3.3.2 Scalar Blinding

Instead of randomizing the base point $\mathcal{P}$, *scalar blinding* deals with the randomization of the second input parameter of *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 Equation 10.

$$k_r = k + r \cdot |\mathcal{E}| \tag{10}$$

Given the blinded scalar k_r, correctness of the underlying scheme is given 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 Equation 11.

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

4. 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, physical and side-channel security and many more. In this section we discuss several optimization strategies and different solution approaches in order to achieve the chosen objectives for our target implementation.

4.1 Implementation Objectives

To the best of our knowledge, there are hardly any published results on hardware implementations targeting ECC beyond the security level of 128 bits. Since the main target for this work is to investigate the suitability of Curve448 in hardware, we designed our architecture to achieve an optimal trade-off in terms of primarily maximizing the performance and throughput while exploring opportunities for resource cost optimization as secondary goal.

4.2 Optimization Objectives

In order to increase the performance of our design to a maximum we applied several optimization strategies which we list in the following.

Register Balancing. It is a common approach to use additional register stages to minimize the critical path delay of hardware architectures. In particular for FPGA implementations this is straightforward since registers are placed after every logic element and would be wasted if they remain unused. We applied manual register balancing to minimize the critical path of each DSP core by using internal register stages in consideration of the latency added by each individual stage.

Parallelization. In particular, hardware implementations can benefit from parallelization techniques which can be applied on various levels of the design hierarchy. We put a particular focus on the parallelization of all required base field operations (addition, subtraction, multiplication) by extensive use of DSP blocks. The placement of multiple core instances on the same device is another opportunity to increase performance, only limited by the available resources of the physical device (simple scalability).

Resource Sharing. Naturally, faster designs and better performance can be purchased by spending more resources. Yet resource sharing may allow reusing existing components instead of implementing additional ones. Similar to parallelization, resource sharing can be applied at different levels of the design hierarchy, i.e., within a single but also among several instances (multi-core design).

Data Path Expansion. For efficient parallel computation, the internal data path of the design has to be expanded in order to provide each component without pipeline stalls. Since this naturally will increase the area utilization of the architecture, we investigated a best trade-off between either a serial and low-area or a parallel and low-latency design.

In general, critical path minimization and parallelization are common approaches to increase the throughput of hardware architectures. We explored several combinations and techniques to achieve the maximum frequency while maintaining a balanced resource utilization in order to allow the implementation of multiple instances in parallel (multi-core design). In the following, we provide a synopsis on our findings and optimization strategies.

4.3 Field Arithmetic Operations

According to Section 3.2.1, 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) which has

[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*.

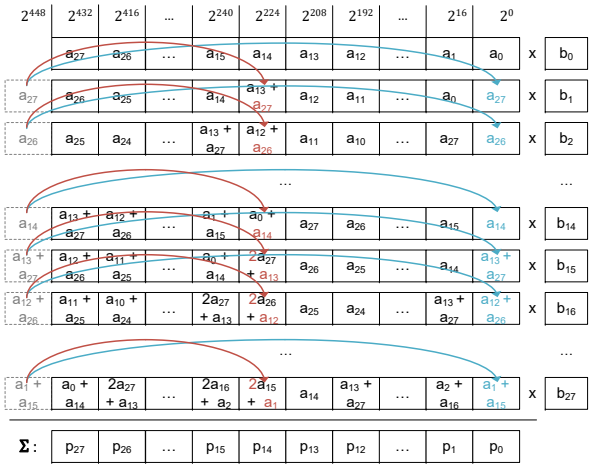

Figure 1: Custom schoolbook multiplication with interleaved reduction

to be designed and optimized thoroughly (cf. Section 5.1) since its operations have a great impact on the overall performance. In this context, we finally achieved better results by entirely manual optimization instead of relying on automated tools. In the following, we explain and discuss all relevant design choice for our Field Arithmetic Unit that is optimized for use with Curve448.

4.3.1 Modular Inversion

In general, hardware designers have to decide whether to perform modular inversion using dedicated units, e.g., based on the binary Extended Euclidean Algorithm (EEA), or to reuse modular multiplication to compute the inversion using FLT or the ISR tweak. We decided against a dedicated unit due to its little performance gain with respect to its area costs (based on a single core architecture). Instead, we decided to perform modular inversion using the ISR tweak as given in Equation 3.

4.3.2 Modular Multiplication

For modular multiplication, we investigated and compared two different approaches: *Karatsuba Multiplication* and *Schoolbook Multiplication with Interleaved Reduction*. Since the primary goal of this work is to explore the suitability and performance of the *point multiplication* for Curve448, 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* in hardware and most important, it maps more naturally to the available hardware resources (DSP units).

4.3.3 Architectural Customizations

In Section 4.2 we prsented a general set of optimization strategies that we considered for the entire design process.

Based on this, we applied manual *register balancing* to minimize the critical path of each DSP core by using internal register stages in consideration of the latency added by each individual stage. Further, registers are balanced between our Arithmetic Core (AC) and Reduction Core (RC) in the same manner.

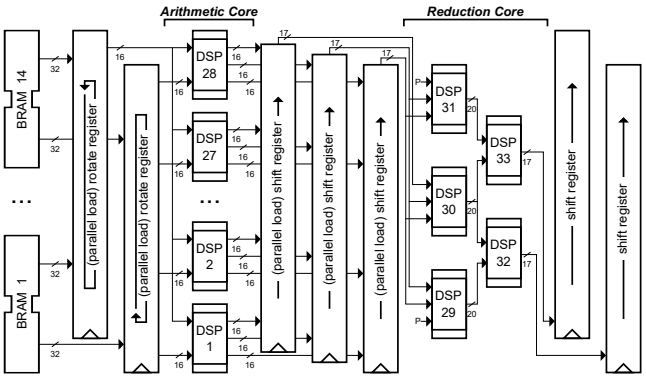

Figure 2: Simplified architecture of the FAU

Parallelization is primarily applied on DSP level by concurrent operation of 28 DSPs that contribute to the computation of all required base field operations (addition, subtraction, multiplication). However, since most of the time the Field Arithemtic Unit (FAU) performs modular multiplications of which each multiplication is performed in 28 steps corresponding to at least 28 cycles, we identified 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 increases latency of modular additions and subtractions as well, since we shar the DSP units also with these operations. Further, *resource sharing* is found on DSP level, since our AC is not only used for modular multiplication but also provides 448-bit additions and subtractions (using an internal carry-save representation).

Eventually, *data path expansion* denotes the use of a full 448-bit data path for memory access as well as for arithmetic operations unlike other designs (cf. Table 2) in order to allow maximum parallelization and optimal performance within all sub-modules. In general, this allows to provide the FAU with all operands on time avoiding or reducing idling phases due to load or store operations.

5. IMPLEMENTATION DETAILS

In this section, we provide implementation details of our Curve448 hardware architecture, in particular the arithmetic units of our design. In addition, we provide details on an enhanced version of our basic architecture 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 into the basic architecture with limited effort and overhead.

5.1 Field Arithemtic Unit

As mentioned earlier, the FAU (as sketched in Figure 2) is the basic component of our architecture. In principle, it can be subdivided into two different components, an AC and a 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 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 Field Arithmetic Unit. Still, both, the AC and the RC, can be operated independently and in parallel in order to alleviate the disadvantage of the serial design.

5.2 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*.

5.2.1 Point Randomization

As explained in Section 3.3.1, *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 and uses a modular multiplication in order to update the x-coordinate. Hence, this countermeasure can be integrated easily, mainly by using an additional multiplication call during the initialization phase of the implementation.

5.2.2 Scalar Blinding

According to Section 3.3.2, the scalar is blinded by a random multiple of the group order and due to the special structure of the underlying prime, it is required to use random factors of at least half of the field size [16]. Hence, the secret scalar is blinded and expanded by 221 bits using two additional DSP units. In particular, we opted to use a blinding factor of $221 = 13 \cdot 17$ bits, since it perfectly fits to the multiplier within the DSP unit that can perform 24×17-bit signed multiplications. However, based on the increased scalar size, the overall performance of the point multiplication is reduced due to additional Montgomery steps that have to be performed.

6. RESULTS

In this section, we present implementation and performance results after place-and-route (PAR) for both designs and provide a comparison to related work. Note, that all results were obtained for a Xilinx XC7Z020CLG484-3 FPGA using the Vivado Design Suite 2015.4.

6.1 Implementation Results

Table 1 provides the implementation results of both designs (including and excluding side-channel countermeasures) in terms of occupied resources. Obviously, the limiting factor for both designs (considering the maximum degree of parallelization for multi-core architectures) is the number of DSP units for the given platform. However, although the resource utilization slightly increases for the protected design, both variants theoretically allow to place up to six cores in parallel (on the chosen mid-range Xilinx Zynq FPGA device).

Table 1: Resource utilization and performance results after place-and-route

	Aspect	Unprotected	SCA-Protected
Single-Core	*Resource Utilization*		
	LUTs	2555 (4.80%)	3583 (6.73%)
	FFs	7049 (6.63%)	7423 (6.98%)
	LSs	1580 (11.88%)	1648 (12.39%)
	DSPs	33 (15.00%)	35 (15.91%)
	BRAMs	14 (10.00%)	14 (10.00%)
	Performance		
	Addition	$11.20ns$/4 cycles	$11.92ns$/4 cycles
	Subtraction	$11.20ns$/4 cycles	$11.20ns$/4 cycles
	Mult.	$89.60ns$/32 cycles	$95.36ns$/32 cycles
	Reduction	$100.80ns$/36 cycles	$107.28ns$/36 cycles
	Mont. Step	$1.84\mu s$/658 cycles	$1.96\mu s$/658 cycles
	Inversion	$92.3\mu s$/32976 cycles	$98.3\mu s$/32976 cycles
	Point Mult.	$0.9ms$/328 286 cycles	$1.4ms$/473 926 cycles

Table 1 also summarizes the performance results for both designs broken down to the group and field arithmetic operations. Since the unprotected design can achieve a higher maximum clock frequency (357 MHz) compared to the protected design (335 MHz), operations are slightly faster although the number of clock cycles are the same for the field arithmetic and group operations. However, due to the increased bit-size of the blinded scalar, the final *point multiplication* of the protected design involves additional iterations of the Montgomery ladder algorithm. This leads to an increase of the overall latency in terms of clock cycles by 45%.

6.2 Comparison

To the best of our knowledge, there are only few implementations of ECC schemes for FPGA devices publicly available that target a security level equal or above 128 bits. Along with the fact that modern FPGA architectures have evolved alot and now include more powerful features compared to earlier generation, a fair and meaningful discussion or comparison of different designs and implementations with previous works 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. Again, we like to emphasize that a simple comparison is not possible.

Table 2 lists several FPGA architectures of ECC schemes over prime fields, including standardized NIST primes (P-224 and P256), Curve25519 (second candidate of RFC 7748) and an implementation of an elliptic curve called Four$\mathbb{Q}$, recently presented at CHES 2016. Note, however, that all these implementations target a lower level of security (112-128 bits) which implies a smaller field size and thus less arithmetic complexity for the basic operations. Still, we would like to highlight some observations from our design in comparison to previous works.

Given the fact, that time complexity is assumed to grow cubic in the field size, we can extrapolate a performance loss of 15-20% in comparison to the implementations listed in Table 2. In this regard, our implementation exceeds these expectations due to its manually optimized architecture, although, slightly more resources are occupied.

Table 2: Comparison of different designs for Elliptic Curve Cryptography over prime fields on FPGAs

Implementation	Security	Device	Architecture				Performance			Ref.
			Data Path	Logic Slices	DSP	BRAM	Cycles	OP/s	MHz	
Four$\mathbb{Q}$										
Single-Core (Mont.)	123 bit	XC7Z020	127 bit	565	16	7	58 967	3222	190	[6]
Single-Core (endo.)	123 bit	XC7Z020	127 bit	1691	27	10	29 739	6389	190	[6]
Multi-Core (endo.)	123 bit	XC7Z020	127 bit	5697	187	110	-	64 730	175	[6]
Curve25519										
Single-Core (unprotected)	127 bit	XC7Z020	34 bit	1029	20	2	79400	2519	200	[14]
Single-Core (protected)	127 bit	XC7Z020	34 bit	1169	22	2	83252	2402	200	[15]
Multi-Core (unprotected)	127 bit	XC7Z020	34 bit	11 277	220	22	34052	32 304	100	[14]
Multi-Core (protected)	127 bit	XC7Z020	34 bit	10 903	220	20	36107	27 695	100	[15]
Curve448										
Single-Core (unprotected)	224 bit	XC7Z020	448 bit	1580	33	14	328 286	1087	357	**new**
Single-Core (protected)	224 bit	XC7Z020	448 bit	1648	35	14	473 926	708	335	**new**

6.3 Curve25519 vs. Curve448 in Hardware

The underlying finite field and its corresponding base field arithmetic is an essential difference of Curve448 compared to Curve25519. Instead of a Pseudo Mersenne prime, Curve448 uses a Solinas prime which results in the need for a more complex reduction scheme. In combination with the significantly larger field size, this required a completely different modular multiplier design to still fully benefit from (additionally required) DSP units. In this context, the wider multiplier design of Curve448 was augmented to completely absorb the modular addition and subtraction operation in its data path, resulting in a single component for all modular operations. As a consequence, this single component design enabled a simple memory architecture on the higher arithmetic level compared to the more complex butterfly architecture of Curve25519 to connect two separate arithmetic components.

7. CONCLUSION

In this work we presented a new architecture for ECC based on Curve448, the elliptic curve for high-security application as defined in RFC 7748. Although Curve448 was mainly designed for efficient implementations on software platforms, we demonstrated that it also perfectly maps to modern FPGA architectures. Further, we showed that our implementation can be augmented with countermeasures against side-channel attacks at moderate costs. Whereas our unprotected allows more than 1000 *point multiplications* per second on a mid-range Xilinx XC7Z020 FPGA, our protected design achieves a performance of about 700 operations per second. Finally, we like to underline that the performance of both single core implementations should be sufficient to match the requirements a large number of high-security applications.

8. REFERENCES

[1] D. J. Bernstein and T. Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography, 2016. https://safecurves.cr.yp.to/.

[2] G. M. de Dormale and J. Quisquater. High-speed hardware implementations of Elliptic Curve Cryptography: A survey. *Journal of Systems Architecture*, 2007.

[3] J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures. In *HOST*, 2010.

[4] T. Güneysu and C. Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In *CHES*, 2008.

[5] M. Hamburg. Ed448-Goldilocks, a new elliptic curve. *IACR Cryptology ePrint Archive*, 2015.

[6] K. Järvinen, A. Miele, R. Azarderakhsh, and P. Longa. Four $\mathbb{Q}$ on FPGA: New Hardware Speed Records for Elliptic Curve Cryptography over Large Prime Characteristic Fields. In *CHES*, 2016.

[7] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 1987.

[8] M. Lochter and J. Merkle. Elliptic curve cryptography (ECC) brainpool standard curves and curve generation. Technical report, 2010.

[9] G. Locke and P. Gallagher. Fips pub 186-3: Digital Signature Standard (DSS). *Federal Information Processing Standards Publication*, 2009.

[10] V. S. Miller. Use of elliptic curves in cryptography. In *CRYPTO*, 1985.

[11] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 1987.

[12] G. Orlando and C. Paar. A Scalable GF(p) Elliptic Curve Processor Architecture for Programmable Hardware. In *CHES*, 2001.

[13] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle. Hardware Implementation of an Elliptic Curve Processor over GF(p). In *ASAP*, 2003.

[14] P. Sasdrich and T. Güneysu. Efficient Elliptic-Curve Cryptography Using Curve25519 on Reconfigurable Devices. In *ARC*, 2014.

[15] P. Sasdrich and T. Güneysu. Implementing Curve25519 for Side-Channel-Protected Elliptic Curve Cryptography. *TRETS*, 2015.

[16] W. Schindler and A. Wiemers. Efficient side-channel attacks on scalar blinding on elliptic curves with special structure. In *NIST Workshop on ECC Standards*, 2015.