Closing the Gap in RFC 7748: Implementing Curve448 in Hardware

Pascal Sasdrich[1], Tim Güneysu[2]

[1] Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany
pascal.sasdrich@rub.de

[2] University of Bremen and DFKI, Germany
tim.gueneysu@uni-bremen.de

Abstract. With the evidence on comprised cryptographic standards in the context of elliptic curves, the IETF TLS working group has issued a request to the IETF Crypto Forum Research Group (CFRG) to recommend new elliptic curves that do not leave a doubt regarding their rigidity or any backdoors. This initiative has recently published RFC 7748 proposing two elliptic curves, known as Curve25519 and Curve448, for use with the next generation of TLS. This choice of elliptic curves was already picked up by the IETF working group *curdle* for adoption in further security protocols, such as DNSSEC. Hence it can be expected that these two curves will become predominant in the Internet and will form one basis for future secure communication.
Unfortunately, both curves were solely designed and optimized for pure software implementation; their implementation in hardware or their physical protection against side-channel attacks were not considered at any time. However, for Curve25519 it has been shown recently that efficient implementations in hardware along with side-channel protection are possible. In this work we aim to close this gap and demonstrate that fortunately the second curve can be efficiently implemented in hardware as well. More precisely, we demonstrate that the high-security Curve448 can be implemented on a Xilinx XC7Z7020 at moderate costs of just 963 logic and 30 DSP slices and performs a scalar multiplication in 2.5ms.

Keywords: RFC7748, Curve448, hardware implementation, FPGA, side-channel protection

1 Introduction

Elliptic Curve Cryptography has emerged to a predominant choice for today's asymmetric cryptography. Cryptosystems such as the ECDSA providing digital signatures or the TLS ciphersuite defining an ephemeral Elliptic Curve Diffie-Hellman Key Agreement are in widespread use worldwide. Unfortunately recent revelations have shown that some cryptographic standards in domain of ECC (to be precise, the NIST Dual_EC_DRBG random number generator [28, 31]) were

deliberately manipulated to enable backdoor access to the randomness generation process for a knowing external party. These doubts about the trustworthiness of the standardized elliptic curves has led to significant efforts to analyze the rigidity of the curve generation process – what finally emerged into a discussion among researchers and standardization bodies to provide and select a next generation of elliptic curves for future cryptography.

In this context, the IETF TLS working group has issued a request to the IRTF Crypto Forum Research Group (CFRG) [15] asking for a recommendation of new elliptic curves, focusing on finding replacements for TLS. The intense discussions within the CFRG have finally led to a selection of elliptic (Edwards) curves for which their rigidity argument was primarily obtained from their optimality for software implementation. Thus, the two elliptic curves Curve25519 and Curve448 have now become part of the RFC 7748 [14] and will form new ciphersuites for the next generation of TLS. Nearly at the same time this choice was picked up by the IETF *curdle* working group for proposing both curves also for further protocols, such as DNSSEC [13]. Hence we can expect that both elliptic curves will become among the most predominant curves in the Internet (and possibly even beyond) and thus will play a central role in our future secure communication.

Unfortunately it should be highlighted that both curves were originally designed to almost exclusively be implemented in software and thus have excluded several other very hardware-related scenarios, such as implementations on embedded devices, hardware platforms and broad considerations about side-channel analysis. In particular, with the advent of the Internet of Things, many embedded hardware devices will certainly need to also support TLS – sometimes even with challenging requirement to provide high throughput. This said, we like to recall the situation with Data Encryption Standard (DES) that was purely optimized for hardware implementation at its design phase and turned out to be rather unsuitable for other (software) platforms. Thus we urgently need to validate the suitability of the two curves as defined in the RFC7748 with respect to their hardware implementation.

Contribution: In this work we present the first implementation of Curve448 on reconfigurable hardware that was formerly proposed by Mike Hamburg as Ed448-Goldilocks [12]. We focus on the implementation and side-channel security of Curve448 since a few aspects of Curve25519 have been recently addressed Sasdrich and Güneysu [26, 27]. Interestingly, we can show that Curve448 has beneficial properties in hardware so that some parts, such as the modular multiplication can be considered as slightly superior to Curve25519ÀAccording to our findings, the single-core implementation achieves a throughput of 400 ECDH scalar multiplications per second on a mid-range Xilinx XC7Z7020 (Zynq) FPGA.

Outline: The remainder of this article is organized as follows: Section 2 summarizes previous work of relevance. In Section 3 we briefly recapitulate the basics of Elliptic Curve Cryptography (ECC) in general and present the mathematical background in particular for the Curve448 elliptic curve. Design rationales of our

implementation are discussed in Section 4 before presenting the final realization on reconfigurable hardware in Section 5. Performance and implementation results are summarized and given in Section 6 along with a detailed comparison to related work. Finally, our work concludes in Section 7.

2 Previous work

Since the introduction by Neal Koblitz [17] and Victor Miller [20], Elliptic Curve Cryptography (ECC) has established as a viable field of research and made its way into a myriad of products. In terms of ECC hardware implementations a wealth of publications exist that cannot all be discussed within the scope of this work. Hence, we would like to restrict our consideration only to the most relevant related works and refer the interested reader to an overview in [5].

Orlando and Paar presented one of the first implementations for ECC in [22] for reconfigurable hardware, applying Montgomery multiplications and precomputations in order to improve performance. Other designs use dedicated multipliers [23] or algorithmic optimizations [25] to increase performance and efficiency. Using integrate Digital Signal Processing abilities of modern FPGAs was initially proposed by Güneysu and Paar [11] and exemplary shown for NIST primes P-224 and P-256. Recently, Sasdrich and Güneysu [26, 27] proposed an implementation that realizes Curve25519 in hardware, the second candidate of the RFC 7748. With respect to physical protection and countermeasures for ECC, Fan et al. provided an excellent survey investigating the different attack vectors such as side-channel analysis and fault injection attacks [9, 10].

3 Background

Although the basic concepts are well known, we briefly introduce the mathematical background of ECC to keep this work self-contained. The section is laid out to put particular emphasis on the specification of Curve448 [12] so that we will detail mostly the necessary background information on group operations and ECDH on Curve448. Note that, although we focus on ECDH in this work, the results can be directly used to derive performance results for ECDSA [3] for digital signatures.

3.1 Elliptic Curve Cryptography

Note that throughout this work, we only consider ECC over $GF(p)$ and exclude other curves explicitly targeting binary or other extension fields.

In general, we can define an elliptic curve over $\mathbb{F}_p = GF(p)$, given its Short Weierstrass equation, as:

$$\mathcal{E}_w : y^2 \equiv x^3 + ax + b \pmod{p}$$

with p prime, $p > 3$ and $a, b \in \mathbb{F}_p$ and satisfying $4a^3 + 27b^2 \neq 0$. Furthermore we can define tuples of affine coordinates (x_i, y_i) as points $\mathcal{P}_i \in \mathcal{E}$ on an elliptic curve $\mathcal{E}$. Considering two points $\mathcal{P}_1, \mathcal{P}_2$ on an elliptic curve $\mathcal{E}$ we can define the basic arithmetic group operation for ECC as *point addition* $\mathcal{P}_3 = \mathcal{P}_1 + \mathcal{P}_2$ using the so called *tangent-and-chord* rule to compute the resulting point $\mathcal{P}_3$. However, the group operation distinguishes two different cases. First, if $\mathcal{P}_1 = \mathcal{P}_2$ this is considered as *point doubling* and second, if $\mathcal{P}_1 \neq \mathcal{P}_2$ this is considered as *point addition*. Note, that both operations use different formulas to perform the group operation. Moreover, by introducing projective coordinate representations defining points on elliptic curves as tuples of (x_i, y_i, z_i), we can relax the elliptic curve equation and relinquish costly modular inversion within the formulas for point addition and doubling at cost of additional multiplications. Hence, in practice we often use projective rather than affine coordinate representation to improve efficiency and performance of implementations.

In order to apply elliptic curves for cryptography, the Elliptic Curve Discrete Logarithm Problem (ECDLP) is defined as the problem to find a scalar k, given two points $\mathcal{P}$ and $\mathcal{Q}$, such that $\mathcal{Q} = k \cdot \mathcal{P}$ holds whereas the *point multiplication* is defined as a sequence of k additions of a base point $\mathcal{P}$. In particular, the ECDLP is the fundamental problem that allows to employ elliptic curves for cryptographic protocols and schemes such as e.g., the Elliptic Curve Diffie-Hellman (ECDH) key exchange [6], the Elliptic Curve Digital Signature Algorithm (ECDSA) [16] or the ElGamal encryption [8].

3.2 Curve448 Specification

Curve448, originally introduced as Ed448-Goldilocks by Mike Hamburg [12] and specifically designed as an alternative to existing NIST [19] (based on Generalized Mersenne Primes) or Brainpool curves [18], targeting the high-end security range between 384 to 521 bits. During the design phase of Curve448 the author followed the SafeCurves policies and criteria [4] and optimized parameters for optimal implementation in software. Technically, Curve448 is an untwisted Edwards Curve [7] defined by following equation:

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

with $d = -39081$ and the p being a Solinas prime of shape $p = 2^{448} - 2^{224} - 1$ with its golden ratio $\phi = 2^{224}$. Due to its design process according to the SafeCurves requirements, this curve aims to facilitate secure implementations by eliminating common vulnerabilities and flaws by construction.

It is striking that the security level of Curve448 with 448 bits is higher than used in many contemporary systems. Nevertheless it offers a high versatility and flexible implementation for a wide range of different software platforms. Due to the fact that $224 = 56 \times 4 = 28 \times 8 = 14 \times 16$, Curve448 has an inherent support for different architectures ranging from 8- to 64-bit processors. In addition, the Solinas prime with its golden ratio ϕ allows efficient Karatsuba-based multiplications for two values $A = (a_0 + a_1\phi), B = (b_0 + b_1\phi)$ using:

Algorithm 1 Single step of Montgomery ladder

Input : $A = -d$, $\mathcal{Q}_1 = (X_1, Z_1)$, $\mathcal{Q}_2(X_2, Z_2)$ and $\mathcal{Q}_3 = Q_1 - Q_2 = (X_3, 1)$
Output : $\mathcal{Q}_4 = 2\mathcal{Q}_1 = (X_4, Z_4)$ and $\mathcal{Q}_5 = Q_1 + Q_2 = (X_5, Z_5)$

1:	$T_1 \leftarrow X_1 + Z_1$	$(X_1 + Z_1)$
2:	$T_2 \leftarrow X_1 - Z_1$	$(X_1 - Z_1)$
3:	$Z_1 \leftarrow X_2 - Z_2$	$(X_2 - Z_2)$
4:	$X_1 \leftarrow T_1 \cdot Z_1$	$(X_1 + Z_1)(X_2 - Z_2)$
5:	$Z_1 \leftarrow X_2 + Z_2$	$(X_2 + Z_2)$
6:	$X_2 \leftarrow T_2 \cdot Z_1$	$(X_1 - Z_1)(X_2 + Z_2)$
7:	$Z_2 \leftarrow X_1 - X_2$	$((X_1 + Z_1)(X_2 - Z_2) - (X_1 - Z_1)(X_2 + Z_2))$
8:	$Z_1 \leftarrow Z_2^2$	$((X_1 + Z_1)(X_2 - Z_2) - (X_1 - Z_1)(X_2 + Z_2))^2$
9:	$Z_5 \leftarrow X_3 \cdot Z_1$	$X_3((X_1 + Z_1)(X_2 - Z_2) - (X_1 - Z_1)(X_2 + Z_2))^2$
10:	$Z_1 \leftarrow X_1 + X_2$	$((X_1 + Z_1)(X_2 - Z_2) + (X_1 - Z_1)(X_2 + Z_2))$
11:	$X_5 \leftarrow Z_1^2$	$((X_1 + Z_1)(X_2 - Z_2) + (X_1 - Z_1)(X_2 + Z_2))^2$
12:	$Z_1 \leftarrow T_1^2$	$(X_1 + Z_1)^2$
13:	$T_1 \leftarrow T_2^2$	$(X_1 - Z_1)^2$
14:	$X_4 \leftarrow Z_1 \cdot T_1$	$(X_1 + Z_1)^2(X_1 - Z_1)^2$
15:	$T_2 \leftarrow Z_1 - T1$	$((X_1 + Z_1)^2 - (X_1 - Z_1)^2)$
16:	$T_1 \leftarrow T_2 \cdot A$	$A((X_1 + Z_1)^2 - (X_1 - Z_1)^2)$
17:	$T_1 \leftarrow T_1 + Z_1$	$(A((X_1 + Z_1)^2 - (X_1 - Z_1)^2) + (X_1 + Z_1)^2)$
18:	$Z_4 \leftarrow T_1 \cdot T_2$	$4X_1Z_1(X_1^2 + AX_1Z_1 + Z_1^2)$

Return : $\mathcal{Q}_4 = (X_4, Z_4)$ and $\mathcal{Q}_5 = (X_5, Z_5)$

$$(a_0 + a_1\phi) \cdot (b_0 + b_1\phi) = (a_0b_0 + a_1b_1) + ((a_0 + a_1)(b_0 + b_1) - a_0b_0)\phi.$$

However, in this work we are going to show that not only software implementations allow fast computations but also implementing Curve448 efficiently in hardware is possible.

3.3 Group Operation of Curve448

Edwards curves can be transformed (bidirectionally) into Montgomery curves and support the application of Montgomery ladder [21] algorithms in order to compute point multiplications. A single step of the Montgomery ladder, as shown in Algorithm 1, computes a combined point addition and point doubling of two points $\mathcal{Q}_1, \mathcal{Q}_2$ provided that the point $\mathcal{Q}_3 = \mathcal{Q}_1 - \mathcal{Q}_2$ is known. In total, each step of the Montgomery ladder involves 4 squaring, 5 general multiplications, 1 constant multiplication, 4 additions and 4 subtractions in $GF(p)$ and can be computed using projective coordinate representations of the points while omitting the y-coordinate. Eventually, each step returns two points $\mathcal{Q}_4 = 2\mathcal{Q}_1$ and $\mathcal{Q}_5 = \mathcal{Q}_1 + \mathcal{Q}_2$ as:

$$\begin{aligned}
X_4 &= (X_1 - Z_1)^2(X_1 + Z_1)^2 \\
Z_4 &= 4X_1Z_1(X_1^2 + AX_1Z_1 + Z_1^2) \\
X_5 &= Z_3((X_1 - Z_1)(X_2 + Z_2) + (X_1 + Z_1)(X_2 - Z_2))^2 \\
Z_5 &= X_3((X_1 - Z_1)(X_2 + Z_2) - (X_1 + Z_1)(X_2 - Z_2))^2
\end{aligned}$$

3.4 ECDH on Curve448

For running the ECDH protocol for key agreement, two point multiplications $\mathcal{Q} = k \cdot \mathcal{P}$ using a publicly known base point $\mathcal{P}$ and a secret scalar k need to be performed. Applying the previously mentioned group operations in terms of single steps of the Montgomery ladder, an entire scalar multiplication on Curve448 can be performed in 448 steps. However, this does not include transformation from affine to projective coordinate representations and back, although it is necessary in practice. Therefore, using the affine coordinates of a base point $\mathcal{P}$, the projective coordinates are initialized as $(X, Y, 1)$. Eventually, after performing the Montgomery ladder using projective coordinates, the result is converted back as $(X \cdot Z^{-1}, Y \cdot Z^{-1})$ using the modular inverse of Z.

However, modular inversions are relatively expensive operations in hardware and thus often performed exploiting Fermat's Little theorem (FLT) to avoid a costly implementation of the Extended Euclidean Algorithm (EEA). Another approach is the following trick to compute inverse square roots in order to find the inverse in $GF(p)$ provided that $p \equiv 3 \ mod \ 4$:

$$\frac{1}{\pm\sqrt{x}} = x^{\frac{p-3}{4}} \quad \text{then} \quad \frac{1}{x} = x \cdot (\frac{1}{\pm\sqrt{x}})^2$$

Hence, one can compute the modular inverse x^{-1} as

$$x^{-1} = x \cdot (x^{\frac{p-3}{4}})^2 = x^{\frac{p-1}{2}} \ mod \ p$$

which is slightly faster than using FLT and still not as expensive as EEA.

4 Rationales for Curve448 in Hardware

In this section, we discuss how we mapped the software-oriented design and parameter selection of Curve448 to an implementation in reconfigurable hardware. Common design strategies for ECC co-processor usually follow a bottom-up approach, separating the architecture into three or four different layers: *Finite Field Arithmetic*, *Elliptic Curve Arithmetic*, *Scalar Multiplication*, and *Protocol Layer (optional)*. We adopt this approach as described in the following.

For the very basic arithmetic layer, we need to briefly review the components of FPGA devices that can support the design high-performance and/or low-area implementation of Curve448. We therefore explain and discuss our picks for the *Finite Field Arithmetic* and *Elliptic Curve Arithmetic* and briefly outline how our choices help to improve resistance against implementation attacks. Our decision taken in this section obviously have a direct consequence in terms of the resource cost and runtime as presented in Section 6.

4.1 FPGA Components

Modern FPGAs are highly sophisticated integrated circuits that provide large programmable facilities to nearly implement any user-defined digital logic circuit. Besides the general-purpose logic resources and the dynamic interconnection network, FPGAs usually come with dedicated memory blocks (BRAM) and

hardcores such as e.g., Digital Signal Processing (DSP) accelerators or high-bandwidth transceivers.

Digital Signal Processing (DSP) Hardcores. DSP hardcores were integrated in recent FPGA families to primarily accelerate signal processing applications. As shown in [30, 11] they can be also used to support multi-precision modular integer arithmetic as required in asymmetric cryptography. Note that DSP hardcores continuously improved over time so that additional arithmetic functions as well as asymmetric data paths are supported in latest FPGA devices.

Each DSP harcore itself can be considered as a run-time configurable processor providing several arithmetic operations including addition and subtraction, multiplication (combined with pre-additions) and accumulation. For modern Xilinx FPGA families, the DSP accelerators support up to 25×18-bit signed multiplications and up to 48-bit wide additions and subtractions. In order to maximize performance and throughput of the accelerators, several pipelining stages can be enabled allowing to operate the DSP cores at maximum device frequency.

Block Memory (BRAM) Instances. Besides fast and efficient computations, many applications have massive memory demands and fast and compact storage is required. For this reason, modern FPGAs provide dedicated low-power and area-optimized BRAM instances. Each BRAM can be configured as a true dual-port memory providing up to 36 Kbits of storage with several configuration alternatives ranging from $1K \times 32$-bit to $32K \times 1$-bit.

4.2 Finite Field Arithmetic

For ECC, *modular addition*, *modular subtraction*, *modular squaring*, *modular multiplication* and *modular inversion* are essential finite field operations required for the basic group operation. Modular addition and subtraction are related and can be realized in a single hardware instance. Similar, modular squaring is usually performed as modular multiplication in hardware to save the resources for an additional arithmetic unit. Besides, modular inversion is the most expensive operation and is realized by building a dedicated arithmetic unit for the Extended Euclidean Algorithm (EEA). Alternatively, by applying Fermat's Little Theorem (FLT) or finding Inverse Square Roots (ISR) it can be implemented as a sequence of multiplications and squarings. Hence, finite field arithmetic can be realized by two different units, one for modular addition and subtraction and another one for modular squarings, multiplications and inversions, which both should be accelerated by DSP hardcores to maximize performance.

First, modular addition and subtraction will always compute $s = a \pm b$ (depending on the selected operation). The result of this operation, given that $0 \leq a, b < p$, can be slightly beyond the bounds of the field and has to be reduced. Hence, a second computation $s' = s \mp p$, is performed. Finally, depending

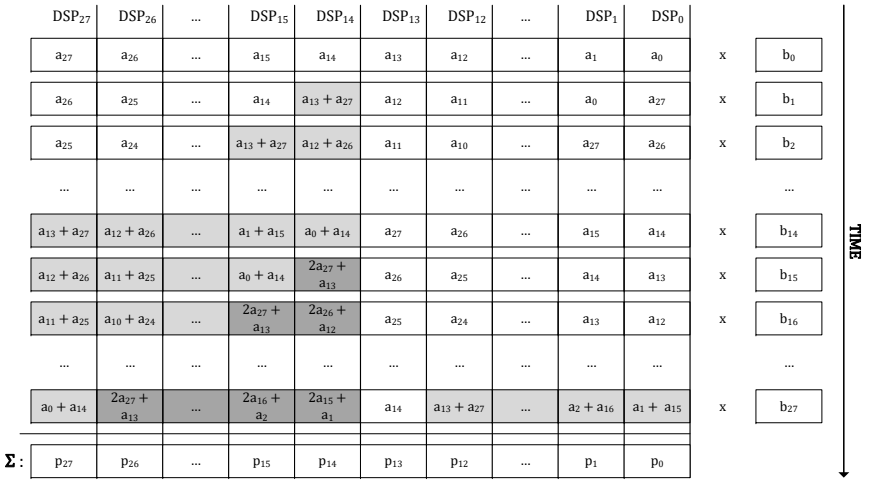

Fig. 1: Parallel modular multiplication with integrated reduction using DSP accelerators

on the last carry of s, the final result is selected i.e., s' is returned if a carry occurred and s otherwise. Both computations can be implemented in parallel using two DSP accelerators which perform the entire modular addition or subtraction sequentially on 32-bit wide operands. Note, that in case of $p \leq s < 2^{448}$ no carry occurs, although the result has to be reduced. However, this can be ignored since a subsequent multiplication still works and applies the required reduction to the result.

Modular multiplication benefits from the Solinas trinomial shape of the Curve448 prime and is performed in two phases. First, partial products are computed in parallel and accumulated using DSP accelerators as 16×16-bit multipliers and accumulators. The partial product computation can be interleaved by an integrated pre-reduction using the Solinas prime shape and the fact that $2^{448} = 2^{224} + 1 \mod p$. In particular, the pre-addition functionality of DSP accelerators and occasional shifting operations can be used to perform the integrated reduction as shown in detail in Figure 1. During a second phase, the DSP accelerators are reused as 32-bit adders to compute the final accumulation of the partial products and determine the entirely reduced result.

4.3 Elliptic Curve Group Arithmetic.

Point addition and point doubling, the basic elliptic curve group operations, can be implemented in several ways and using different formulas, depending e.g., on the chosen coordinate representation of the points. In its most simple form, points on elliptic curves are expressed in affine coordinates as a tuple (x, y) which

allows to compute the group operations using the *tangent-and-chord* rule. However, in order to avoid expensive modular inversions during computation, points are usually transformed into projective coordinate representations (X, Y, Z) for which appropriately modified point addition and doubling formulas need to be applied.

In our implementation, we will use reduced projective coordinates (X, Z) for internal point representation along with a Montgomery ladder to perform point additions and doublings. Reduced projective coordinates avoid the additional use of the Y coordinate in order to reduce memory requirements. However, knowing the elliptic curve equation $\mathcal{E}_d$ and X, the Y coordinate can be entirely recovered (except for one bit for the sign). Montgomery ladders for reduced projective coordinates can be implemented in constant time using 10 modular multiplications and 8 modular additions and subtractions per step (as shown in Algorithm 1). Each step of the Montgomery ladder performs combined point addition and point doubling and is executed $log_2(k)$ times (depending on the bit-size of the scalar k).

4.4 Countermeasures against Physical Attacks

The proposals Curve25519 and Curve448 have been developed for software implementation where solely the cache/timing side channel was considered as a relevant physical threat that should be countered by careful timing-invariant implementation. For embedded systems, however, this physical threat model is typically insufficient since many other side channels (e.g., power, electromagnetic emanations, etc.) may be present and useful for a physical attacker.

As mentioned before, our implementation performs per se time-invariant scalar multiplications due to the application of the Montgomery ladder and the constant-time design for basic field arithmetic operations such as modular addition/subtraction and modular multiplication. In addition, since each Montgomery step function always computes the same operations, only operands are swapped, inherent resistance against implementation attacks like Simple Power Analysis (SPA) is given.

As discussed in [9, 10], further protection mechanisms need to be applied to defeat more sophisticated side-channel attacks, such as Correlation and Differential Power Analysis (CPA/DPA). This is optimally achieved by combining several arithmetic countermeasures like scalar blinding, projective coordinate randomization or implementation techniques like memory address scrambling. Certainly, all these countermeasure come with an overhead and additional cost in terms of area and performance.

We therefore develop a basic core that achieves minimum protection against timing and SPA attacks by using Montgomery ladder for scalar multiplication. In a second step, we will add a selection of additional advanced countermeasures to improve resistance against DPA/CPA attacks. In the following sections we present details on the type of additional countermeasures and their associated cost in terms of area and performance.

5 Implementation Details

In this section, we provide implementation details of our single core instance of the Curve448 processor. For comparison reasons with an existing Curve25519 implementation we chose a mid-range Xilinx XC7Z7020 (Zynq) FPGA as target platform but we like to highlight that the core can be easily ported to Spartan-6 or UltraScale FPGAs as well. We conclude this section with a brief outline about results for a high-performance multi-core architecture based on our Curve448 core.

5.1 Arithmetic Units

Centerpiece of the ECC co-processor are the two finite field arithmetic units to compute modular additions/subtractions and modular multiplications. Both arithmetic units are implemented using the optimal amount of DSP accelerators[3] in order to increase efficiency and throughput of the components.Note that the remaining field operations, i.e., modular squarings and modular inversions, are implemented by reusing the modular multiplication core.

Modular Addition and Subtraction. In the previous section we already explained the basic idea for computing additions and subtractions in $GF(p)$ by always subtracting or adding the prime to the sum or difference and checking whether the reduced result is correct or not. Figure 2 shows the internal construction of the modular addition and subtraction unit making use of two DSP accelerators. Depending on an external selection signal, both DSPs can change their operation mode (addition or subtraction) during runtime. Although all DSPs support additions and subtractions for up to 48-bit wide operands, we decided to implement only 32-bit operations for compatibility reasons with the memory interface. An entire addition or subtraction of two field elements with 448-bit size is computed sequentially and requires 14 clock cycles. However, most additions and subtractions can be computed in parallel to concurrently running multiplications which allows to hide most of the computation times. In total, the addition and subtraction can be implemented in 5 slices (6 LUT, 6 FF) and 2 DSP hardcores.

Modular Multiplication. The modular multiplication mainly is constructed by 28 DSPs and takes 28 clock cycles to compute and accumulate partial products (with integrated reduction as described in Section 4) and computes and reduces the final result within another 4 clock cycles. Figure 3 presents the architectural details of the modular multiplication unit.

Before any multiplication operation can be started, both operands a, b have to be loaded into provided shift registers. Although the multiplication internally

[3] A more detailed discussion of the optimal number of DSPs and several design options can be found in Section 6.5.

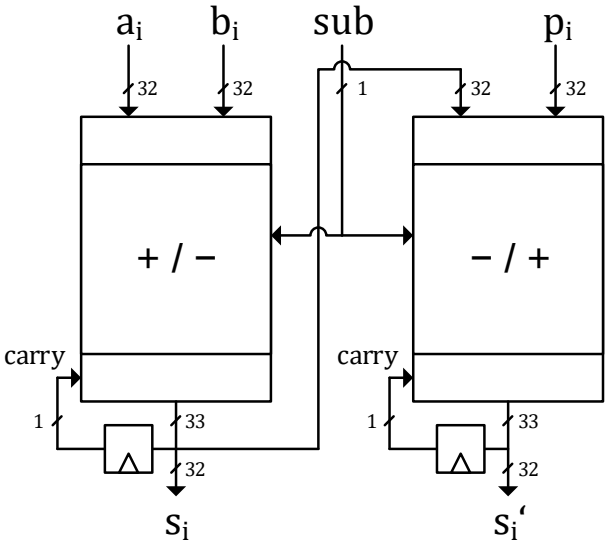

Fig. 2: Modular addition and subtraction based on two DSP hardcores.

operates on 16-bit values, loading (and storing) can be done in 14 clock cycles using 32-bit values to increase performance and ensure compatibility with the memory interfaces. Note that, while storing a result c into attached BRAMs, new operands can be loaded in parallel. Thus, only the first multiplication result requires full 60 clock cycles, each subsequently computed result is available and stored after another 46 clock cycles.

In total, modular multiplication can be implemented in 764 slices (2737 LUT, 1356 FF) and 28 DSP accelerators. All DSP cores provide two different configurations that can be exchanged during runtime to allow reutilization of the multipliers as final accumulation and addition stage.

Inversion. As mentioned before, inversion is realized as a sequence of multiplications and squarings. In order to increase performance and decrease latency, results of previous multiplications do not have to be stored into the BRAM but can be fed back directly to the operand shift registers using multiplexers.

In total, modular inversion can be implemented by 459 multiplications and squarings and requires 21 699 clock cycles. Hence, direct feedback of results allows to save at least 459 clock cycles or 2% of computation time.

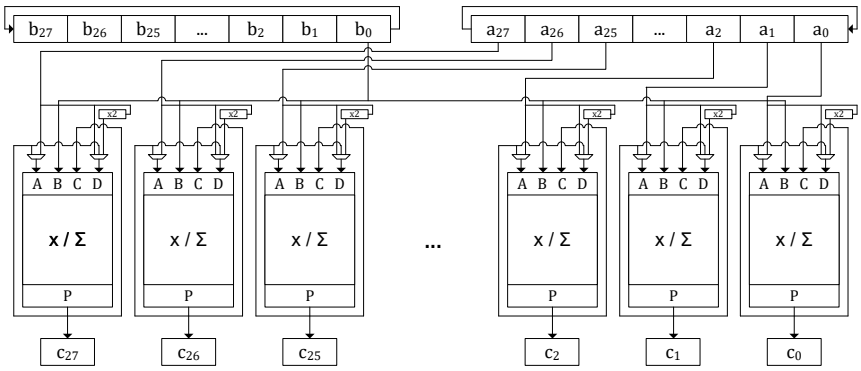

Fig. 3: Modular multiplication using DSP accelerators.

5.2 Co-Processor Architecture

Inspired by the fact that virtually every addition or subtraction during a single Montgomery ladder step is followed by a multiplication and vice versa, we decided to implement our global co-processor architecture as depicted in Figure 4 for interleaving operations in a straightforward but highly efficient way. The two BRAM primitives and both arithmetic cores are connected in butterfly configuration what enables efficient and parallel operation of both arithmetic cores. Note, however, that in rare cases multiplications are also directly followed by multiplications. In order to provide the correct BRAM with necessary operands, dummy additions (i.e., with one operand equal to zero) are used to fit the aforementioned computational scheme.

In total, 448 steps of the Montgomery ladder are executed, each computing a combined point addition and doubling. Each step intrinsically consist of a sequence of 10 multiplications and 8 additions or subtractions. Due to data dependencies and latencies of the BRAM primitives, a single step requires 506 clock cycles in total so that the entire Montgomery ladder is performed in 227 882 clock cycles.

5.3 Side-Channel Protection and Countermeasures

As indicated by the results from [9, 10], we decided to implement a selection of three different countermeasures to harden our design against advanced side-channel attacks. To ensure a correct and safe operation of all countermeasures, we need to add an external (pseudo) random number generator (RNG) that can provide at minimum 573 bits of entropy per scalar multiplication operation. As an option, the randomness can be provided by a suitable block cipher (for example, such as AES) that is used as randomness provider. Although this is certainly an essential component of the protected design, we will not discuss this in further detail.

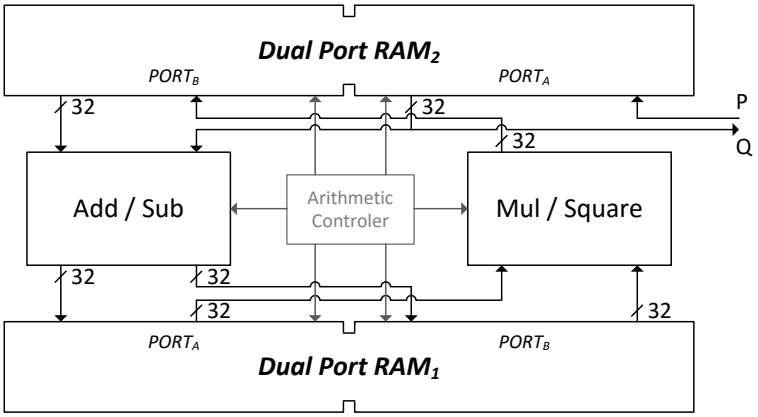

Fig. 4: Curve448 core architecture

Scalar Blinding. A common countermeasure that we also adopt is scalar blinding to randomize the secret scalar using the group order $\#\mathcal{E}$ of Curve448. In particular, the secret scalar k is rewritten as

$$\bar{k} = k + b \cdot \#\mathcal{E}$$

where b is a random blinding factor of size of around 128 bit [29]. Due to performance reasons and the limitation of the DSP hardcores to process only a maximum of 24-bit at a time, we opted to use marginally smaller blinding factors of 120 bit. Nevertheless we do not expect that this slight reduction has a noticeable impact on the overall security of our design.

Coordinate Randomization. Since we employ projective coordinates, this allows to randomize the base point and subsequent intermediate results. Therefore, we chose a random co-factor λ and update our base point $\mathcal{P} = (X, Z)$ (with $Z = 1$) as

$$\lambda\mathcal{P} = (\lambda X, \lambda Z) = (\lambda X, \lambda)$$

Fortunately, we ca reuse our modular multiplication unit for this countermeasure. Since Z has a bit length of 448 bit, this countermeasure requires the major portion of the supplied randomness.

Address Shuffling. Although the application of the Montgomery ladder avoids key dependent operations by performing point doubling and point addition operations in any case, it still has some weaknesses that might be exploited for DPA attacks. Due to the fact, that inputs are swapped depending on the state

of current scalar bit, this can be detected in the BRAM addressing. In order to avoid successful attacks, our design chooses a random 5-bit offset and shuffles internal addresses for every scalar multiplication. Note that it is also possible to shuffle addresses at any single step of the Montgomery ladder, but this certainly comes at the expense of additionally required randomness and higher latency.

5.4 Multi-Core Architecture

For high-performance applications such as TLS hardware accelerators, it is desirable to maximize the overall throughput by using parallelization techniques. Of course, the degree of parallelization is limited by the underlying FPGA devices and its provided resources. For instance, using a mid-range FPGA like Xilinx XC7Z7020 the limiting factor is the number of available DSP accelerators which would allow to implement 7 single-core instances (or 6 including side-channel countermeasures) in parallel. Note that for such a scenario, it might be interesting to implement a dedicated inversion unit which is shared between all cores or offers the functionality of batch inversion. Since inversions account for nearly 10% of computation time, a dedicated inversion unit would also allow to increase efficiency and throughput of the overall design. Note that we did not explicitly implement a multi-core architecture due to the fact that it is highly application- and device-specific. Yet we provide estimates on the area and performance in Section 6 to allow an assessment on the expected throughput for a medium-cost FPGA device (and to allow a comparison to other related works).

6 Results

All implementation results were obtained using Xilinx Vivado 2015.4 after place-and-route. In this section, we present the implementation and performance results of our design and compare it to existing work.

6.1 Implementation Results

Table 1 summarizes of the resource consumption for our single-core Curve448 co-processor implemented on a Xilinx XC7Z7020 FPGA. In general, our design occupies 7% of the general purpose logic in terms of slices what breaks down to 6% of the Look-Up Tables (LUT) and 2% of the Flip-Flops (FF). Furthermore 30 DSP hardcores (14%) and 2 BRAMs (1%) are consumed by a single instance of our co-processor. Obviously, the number of available DSP hardcores is the limiting factor with respect to further parallelization and multi-core instantiations of our design. We still like to emphasize that our architecture has very moderate requirements in terms of area consumption while providing an extremely high security level. It seems that it is easily possible to place and run a variety of different applications in parallel to our ECC co-processor (such as other cryptographic components or even the full TLS stack), even on low- and mid-range FPGAs.

Table 1: Resource consumption after place-and-route on a Xilinx XC7Z7020.

Resource	**Used**	**Available**	**Utilization**
Number of Look-Up Tables	3360	53 200	6.32%
Number of Flip-Flops	1891	106 400	1.78%
Number of occupied Slices	963	13 300	7.24%
Number of DSP48E1	30	220	13.64%
Number of RAMB36E1	2	140	1.43%

Table 2: Performance of the single-core architecture on a Xilinx XC7Z7020.

Operation	**Cycles**	**Time**
Addition *(in $GF(p)$*	14	140 ns
Subtraction *(in $GF(p)$)*	14	140 ns
Multiplication *(in $GF(p)$)*	32	320 ns
Montgomery Step	506	5 μs
Montgomery Ladder	227 882	2.3 ms
Inversion *(in $GF(p)$)*	21 699	0.2 ms
Scalar Multiplication *($k \times \mathcal{P}$)*	249 581	2.5 ms

6.2 Performance Results

In Table 2 we provide timing and performance results highlighting different aspects of our design. In total, our single-core architecture can be operated at a maximum frequency of 100 MHz, due to the critical path of the modular multiplication unit, and thus can perform about 400 ECDH operations (scalar multiplications) per second.

As mentioned before, the most time consuming operation is the modular multiplication that takes 32 cycles (excluding BRAM access). Since each modular addition/subtraction takes 14 clock cycles, two additions or subtractions can be perfectly executed in parallel. Hence, the performance of a single step of the Montgomery ladder is mainly dependent of the timing of the modular multiplication. Due to further timing and data dependencies the entire step executes within 506 clock cycles. The total Montgomery ladder, combining 448 calls of the step function, requires about 228 000 clock cycles and is finalized by a modular inversion. The inversion requires about 22 000 clock cycles and is eventually required for the back transformation of the result from projective coordinates to affine coordinates. In total, a full scalar multiplication requires about 250 000 clock cycles, or 2.5ms at a frequency of 100 MHz.

6.3 Side-Channel Countermeasures

Table 3 summarizes the results of our augmented Curve448 single-core that provides additional side-channel protection. Obviously, the address scrambling is

Table 3: Performance and implementation overhead for the side-channel protected design obtained for a Xilinx XC7Z7020.

Countermeasure	**Latency**		**Resources**			**Rand.**
	Initialization	*Operation*	*LUT*	*FF*	*DSP*	*Bits*
Scalar Blinding	197	61040	954	741	2	120
Coordinate Randomization	47	-	-	4	-	448
Address Shuffling	-	-	124	3	-	5

implemented at virtually no additional cost. On the other side, scalar blinding is the most costly countermeasure in particular in terms of performance loss, mainly due 120 additional steps of the Montgomery ladder. Despite the lower performance, this countermeasure also requires 2 additional DSPs as well as significant portion of the general purpose logic. However, in terms of randomness, only 120 bits of fresh randomness per ECDH computation are necessary. In contrast to this, randomization of the projective coordinate representation slows down the overall performance only by a single modular multiplication but therefore requires 448 bit of fresh randomness per scalar multiplication.

However, only the interaction of all countermeasures ensures an adequate protection. Considering each countermeasure at its own might improve performance or reduce area consumption but certainly will result in weaker side-channel resistance.

6.4 Comparison

To the best of our knowledge, this is the first implementation of Curve448 (and any curve with 448-bit parameters) in hardware. Only very few implementations that target a security level above 256-bit are publicly available. This makes a fair and accurate comparison rather difficult. Note, that even the evolution and newer generations of FPGAs prevents a fair and meaningful discussion of the results. However, we still would like to place our results in the context of existing works of ECC co-processors and give an overview on related work – still with the word of warning that straight comparisons are not possible.

In [2] and [1], two designs implementing standardized NIST primes on newer Virtex-4 and Virtex-6 devices are reported. In particular performance results for NIST P-384 and P-521 are of interest since Curve448 provides a similar security level. Note, that the presented architectures have a generic design and allow a flexible support for all NIST primes, thus it would be unfair to compare it directly to our highly optimized architecture. Still, however, we would like to emphasize the obviously moderate resource requirements of our design, confirming the high level of optimization and efficiency providing decent performance results.

It can be also observed that most implementations used in todays applications mostly target prime fields with size of up to around 256-bit, such as [11, 26, 27, 24]. Assuming the time complexity to grow cubic in the field size, we would

Table 4: Comparison of different designs for ECC over $GF(p)$ on FPGAs, for SCA: - means no countermeasures, + only SPA protection, ++ full protection against SPA and DPA, results marked with * are extrapolated

Device	Scheme	Implementation	Performance	SCA	Ref.
XC4VFX12	224-bit	1580 LS/26 DSP	2739 OP/s, 487MHz	-	[11]
XC4VFX12	256-bit	1715 LS/32 DSP	2020 OP/s, 490MHz	-	[11]
XC4VFX12	224-bit	24452 LS/468 DSP	37735 OP/s, 372MHz	-	[11]
XC4VFX12	256-bit	24574 LS/512 DSP	24691 OP/s, 375MHz	-	[11]
XC7Z7020	255-bit	1029 LS/20 DSP	2519 OP/s, 200MHz	+	[26]
XC7Z7020	255-bit	1169 LS/22 DSP	2402 OP/s, 200MHz	++	[27]
XC7Z7020	255-bit	11277 LS/220 DSP	32304 OP/s, 100MHz	+	[26]
XC7Z7020	255-bit	10903 LS/220 DSP	27695 OP/s, 100MHz	++	[27]
XC5VLX330	256-bit	4505 LS/16 DSP	1754 OP/s, 125MHz	-	[24]
XC4VFX100	192-bit	20793 LS/32 DSP	238 OP/s, 60MHz	-	[2]
XC4VFX100	224-bit	20793 LS/32 DSP	196 OP/s, 60MHz	-	[2]
XC4VFX100	256-bit	20793 LS/32 DSP	163 OP/s, 60MHz	-	[2]
XC4VFX100	384-bit	20793 LS/32 DSP	57 OP/s, 60MHz	-	[2]
XC4VFX100	521-bit	20793 LS/32 DSP	25 OP/s, 60MHz	-	[2]
XC6VLX760	192-bit	11200 LS/289 DSP	3333 OP/s, 100MHz	-	[1]
XC6VLX760	224-bit	11200 LS/289 DSP	2857 OP/s, 100MHz	-	[1]
XC6VLX760	256-bit	11200 LS/289 DSP	2500 OP/s, 100MHz	-	[1]
XC6VLX760	384-bit	11200 LS/289 DSP	847 OP/s, 100MHz	-	[1]
XC6VLX760	521-bit	11200 LS/289 DSP	625 OP/s, 100MHz	-	[1]
XC7Z7020	448-bit	963 LS/30 DSP	400 OP/s, 100MHz	+	new
XC7Z7020	448-bit	1146 LS/32 DSP	322 OP/s, 100MHz	++	new
XC7Z7020	448-bit	6750* LS/210* DSP	2800* OP/s, 100MHz	+	new
XC7Z7020	448-bit	6900* LS/192* DSP	1932* OP/s, 100MHz	++	new

expect a performance loss of about factor 5. According to Table 4 this performance factor is indeed around 5-6 which implies that Curve448 can be at least as good implemented in hardware as any other (highly optimized) curves.

Eventually, we can state that our design provides performance that should match the need of most application while preserving a moderate resource footprint which is in terms of CLBs even smaller than that presented in [26]. Besides, for dedicated resources such as DSP hardcores, we can report an additional overhead of only 50% although field size is almost doubled. We attribute this mainly to the straightforward reduction scheme of Curve448 that allows a slightly more efficient hardware implementation.

6.5 Discussion

Although the general design of Curve25519 (as implemented in [11]) and Curve448 is similar, we would like to highlight some fundamental differences

that require different design choices. In particular the arithmetic unit that performs the modular multiplication should be optimized for the underlying prime of Curve448.

For that reason, our multiplier implementation is superior to the multiplier implemented for Curve25519 since multiplication takes only 32 clock cycles while authors in [11] report 34 clock cycles for their implementation of the (smaller) Curve25519. This is interesting since Curve448 has almost twice the bit size of Curve25519, but can be attributed to modified operation interleaving and an expense of 10 additional DSP hardcores. Still the authors employ pipelining for their implementation of Curve25519 what again increases the throughput of each Montgomery step dramatically. Applying pipelining to our multiplier design, however, would results in a gain of only 4 clock cycles per multiplication (28 cycles rather than 32) at the cost of two additional DSPs. Hence, we decided to not implement pipelining, but instead reuse the DSP hardcore as wide adder.

Yet our design is complex and uses several (unregistered) multiplexers. For future work it might be interesting to investigate register balancing to improve the critical path inside the multiplication unit.

7 Conclusion

In this work we present the first hardware implementation of Curve448. Curve448 has become recently part of RFC 7748 is thus recommended for use in future ciphersuites of TLS and other protocols. Although Curve448 was designed for being highly efficient in software, we demonstrated that it can be similarly efficient mapped onto reconfigurable hardware. In addition to that we provided results regarding the cost of additional countermeasures thwarting side-channel analysis. Yet our single-core implementation achieves a throughput of 400 ECDH scalar multiplications per second on Xilinx XC7Z7020 (Zynq) what we expect to be sufficient for most practical applications.

References

1. H. Alrimeih and D. N. Rakhmatov. Fast and flexible hardware support for ECC over multiple standard prime fields. *IEEE Trans. VLSI Syst.*, 22(12):2661–2674, 2014.
2. K. Ananyi, H. Alrimeih, and D. N. Rakhmatov. Flexible hardware processor for elliptic curve cryptography over NIST prime fields. *IEEE Trans. VLSI Syst.*, 17(8):1099–1112, 2009.
3. D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang. High-speed high-security signatures. *J. Cryptographic Engineering*, 2(2):77–89, 2012.
4. D. J. Bernstein and T. Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography, 2016. `https://safecurves.cr.yp.to/`.
5. G. M. de Dormale and J. Quisquater. High-speed hardware implementations of Elliptic Curve Cryptography: A survey. *Journal of Systems Architecture*, 53(2-3):72–84, 2007.

6. W. Diffie and M. E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.
7. H. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007.
8. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1984.
9. 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 J. Plusquellic and K. Mai, editors, *HOST 2010, Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 13-14 June 2010, Anaheim Convention Center, California, USA*, pages 76–87. IEEE Computer Society, 2010.
10. J. Fan and I. Verbauwhede. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In D. Naccache, editor, *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*, pages 265–282. Springer, 2012.
11. T. Güneysu and C. Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 2008.
12. M. Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. `http://eprint.iacr.org/`.
13. IETF. CURves, Deprecating and a Little more Encryption (curdle), 2016. `https://datatracker.ietf.org/wg/curdle/charter/`.
14. IETF. Elliptic Curves for Security, Jan 2016. `https://tools.ietf.org/html/rfc7748`.
15. IRTF. Crypto Forum Research Group (CFRG), 2016. `https://irtf.org/cfrg`.
16. D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.
17. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.
18. M. Lochter and J. Merkle. Elliptic curve cryptography (ECC) brainpool standard curves and curve generation. Technical report, 2010.
19. G. Locke and P. Gallagher. Fips pub 186-3: Digital Signature Standard (DSS). *Federal Information Processing Standards Publication*, 3:186–3, 2009.
20. V. S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology-CRYPTO85 Proceedings*, pages 417–426. Springer, 1985.
21. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.
22. G. Orlando and C. Paar. A Scalable GF(p) Elliptic Curve Processor Architecture for Programmable Hardware. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2001.
23. S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle. Hardware Implementation of an Elliptic Curve Processor over GF(p). In *14th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP 2003), 24-26*

June 2003, The Hague, The Netherlands, pages 433–443. IEEE Computer Society, 2003.
24. D. B. Roy, D. Mukhopadhyay, M. Izumi, and J. Takahashi. 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*, pages 177:1–177:6. ACM, 2014.
25. K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede. Reconfigurable Modular Arithmetic Logic Unit for High-Performance Public-Key Cryptosystems. In K. Bertels, J. M. P. Cardoso, and S. Vassiliadis, editors, *Reconfigurable Computing: Architectures and Applications, Second International Workshop, ARC 2006, Delft, The Netherlands, March 1-3, 2006, Revised Selected Papers*, volume 3985 of *Lecture Notes in Computer Science*, pages 347–357. Springer, 2006.
26. P. Sasdrich and T. Güneysu. Efficient Elliptic-Curve Cryptography Using Curve25519 on Reconfigurable Devices. In D. Goehringer, M. D. Santambrogio, J. M. P. Cardoso, and K. Bertels, editors, *Reconfigurable Computing: Architectures, Tools, and Applications - 10th International Symposium, ARC 2014, Vilamoura, Portugal, April 14-16, 2014. Proceedings*, volume 8405 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2014.
27. P. Sasdrich and T. Güneysu. Implementing Curve25519 for Side-Channel-Protected Elliptic Curve Cryptography. *TRETS*, 9(1):3, 2015.
28. B. Schneier. The Strange Story of Dual_EC_DRBG. `https://www.schneier.com/blog/archives/2007/11/the_strange_sto.html`.
29. N. P. Smart, E. Oswald, and D. Page. Randomised representations. *IET Information Security*, 2(2):19–27, 2008.
30. D. Suzuki. How to maximize the potential of FPGA resources for modular exponentiation. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2007.
31. Wired. RSA Tells Its Developer Customers: Stop Using NSA-Linked Algorithm, Sept 2013. `http://www.wired.com/2013/09/rsa-advisory-nsa-algorithm/`.