FPGA Design of Self-Certified Signature Verification on Koblitz Curves*

Kimmo Järvinen, Juha Forsten, and Jorma Skyttä

Helsinki University of Technology
Signal Processing Laboratory
Otakaari 5A, FIN-02150, Espoo, Finland
(kimmo.jarvinen,juha.forsten,jorma.skytta)@tkk.fi

Abstract. Elliptic curve signature schemes offer shorter signatures compared to other methods and a family of curves called Koblitz curves can be used for reducing the cost of signing and verification. This paper presents an FPGA implementation designed specifically for rapid verification of self-certified identity based signatures using Koblitz curves. Verification requires computation of three elliptic curve point multiplications which are computed efficiently with 3-term multiple point multiplication and joint sparse form. Certain improvements to precomputations associated with multiple point multiplications are introduced. It is shown that, when using parallel processors, it is possible to gain considerable increases in the number of operations per second by allowing slightly longer computation times for single operations. It is demonstrated that up to 166,000 verifications per second can be computed using a single Altera Stratix II FPGA.

1 Introduction

Research on hardware realization of cryptographic algorithms has been intensive during the past few years. Implementation of elliptic curve cryptosystems by using field programmable gate arrays (FPGAs) has been one of the most active areas in the field, and numerous designs have been described in the literature. This paper extends the research on the subject by describing a very efficient implementation designed specifically for one of the most computationally demanding tasks of modern cryptosystems; namely, signature verification.

Elliptic curve cryptography [1, 2] is a branch of public-key cryptography which has recently been a subject of much interest because a high level of cryptographic security is achievable with shorter key lengths than with other existing methods. The implementation computes elliptic curve operations involved in verification of self-certified identity based signatures based on Nyberg-Rueppel signature scheme [3]. The implementation uses one of the standardized Koblitz curves listed in [4], henceforth referred to as the NIST curve K-163, because

* This research was conducted within the Packet Level Authentication (PLA) project at Helsinki University of Technology (TKK). The PLA project is funded by TEKES.

computations are much faster on Koblitz curves [5]. Further improvements in performance are achieved by computing all operations required in signature verification simultaneously by using multiple point multiplication techniques. Performance is increased by introducing certain improvements to precomputations.

Signature verification is a basic operation in many cryptosystems. Applications, such as the Packet Level Authentication (PLA) scheme [6, 7] where computational requirements for signature verifications are very high, directly benefit from the results presented in this paper.

The contributions of the paper include the following:

- Unified point addition and subtraction formulae are presented which can be used in speeding up precomputations in various methods including multiple point multiplications and combings.
- A new algorithm for 3-term joint sparse form precomputations is presented resulting in a major speed up compared to existing methods.
- To the authors' knowledge, this is the first publication where computation time vs. the number of operations per second (ops) tradeoff is being explored when using parallel processing in elliptic curve operations. It is shown that allowing slightly longer latencies can result in considerable increases in ops.
- A highly efficient implementation which utilizes parallel processing is presented for an Altera Stratix II FPGA. The implementation is capable of performing up to 166,000 verifications per second which exceeds all previously presented implementations.
- It is shown that schemes, such as PLA [6, 7], could be feasible if the implementation presented in this paper is used for accelerating verifications.

The remainder of the paper is organized as follows. Sec. 2 presents the preliminaries of elliptic curve cryptography and self-certified identity based signatures. Algorithms that are used in the implementation are introduced and derived in Sec. 3. The implementation is presented and the results are analyzed in Secs. 4 and 5, respectively. Conclusions are drawn in Sec. 6 and the paper ends with certain suggestions of possible directions for the future research.

2 Preliminaries

2.1 Packet Level Authentication

Packet Level Authentication (PLA) is a scheme where the authenticity of packets in IP (Internet Protocol) traffic is verified by signing and verifying them with cryptographic signatures. The authenticity of packets is verified from node to node instead of from point to point as in other schemes. This helps in preventing many threats including denial-of-service (DoS) attacks but as a downside PLA adds the length of the packet header and most importantly is computationally very demanding. Thus, hardware acceleration is essential. [6, 7]

PLA is one of the possible applications for the implementation of the paper as mentioned above and the rationales behind many design decisions originate from the requirements of PLA.

The use of signatures based on elliptic curves instead of other techniques such as RSA or ElGamal is practically mandatory because the length of signatures must be kept in minimum in order to minimize the overhead caused by PLA [6]. Koblitz curves were chosen in order to maximize the speed of the implementation because operations are notably faster on Koblitz curves than on general curves [5]. Self-certified identity based signatures were selected because they result in shorter signatures and reduced computational complexity [8].

Preliminaries of elliptic curve cryptography and self-certified identity based signatures are presented next in Secs. 2.2 and 2.3, respectively.

2.2 Elliptic Curve Cryptography

Every elliptic curve cryptosystem is based on an operation called *elliptic curve point multiplication*, and it is defined as

$$Q = kP$$

where Q and P are points on an elliptic curve and k is an integer.

Koblitz curves [5] are a family of elliptic curves of the form

$$E_K : y^2 + xy = x^3 + ax^2 + 1$$

where $a \in \{0, 1\}$ and x and y are elements of the *finite field* $\mathbb{F}_{2^m}$. Elliptic curve point multiplication is computed with successive *point additions* and *point doublings* with the *binary method* so that, when $k = \sum_{i=0}^{\ell-1} \kappa_i 2^i$, point doublings are performed for all κ_i and point additions when $\kappa_i = 1$. On Koblitz curves, however, point doublings are replaced by computationally cheap *Frobenius maps* which results in significant improvement in performance. Before this feature can be utilized, k needs to be converted into τ-adic representation. Algorithms for finding *τ-adic non-adjacent form* (τNAF) were presented in [9]. When k is represented in τNAF, it has the form $k = \sum_{i=0}^{\ell-1} \kappa_i \tau^i$ where $\tau = \left((-1)^{1-a} + \sqrt{-7}\right)/2$ and $\kappa_i \in \{0, \pm 1\}$ so that $\kappa_i \kappa_{i+1} = 0$ for all i. The average number of non-zero terms in k is $\ell/3$ [9]. Because point additions are required when $\kappa_i \neq 0$ and $\ell \approx m$, point multiplication on E_K requires on average $m/3$ point additions and m Frobenius maps.

A sum of integer multiples of two points, i.e. $k_1 P_1 + k_2 P_2$, can be accelerated with Shamir's trick [10] where the integers are represented as a matrix having k_1 and k_2 as rows. First, $P_1 + P_2$ is precomputed. Point multiplication is carried out with the binary method so that one adds the point P_1 if the column is $\substack{1 \\ 0}$, the point P_2 if $\substack{0 \\ 1}$ and the precomputed point $P_1 + P_2$ if $\substack{1 \\ 1}$. When the column is $\substack{0 \\ 0}$, only point doubling or Frobenius map is performed. When k_1 and k_2 are in NAF, also the point $P_1 - P_2$ is precomputed. Two integers can be represented in *joint sparse form* (JSF) [11] in order to maximize the number of zero columns. JSF was generalized for n integers in [12]. JSF can be used also for Koblitz curves as an algorithm for finding τ-adic JSF (τJSF) for two integers was presented in [13]. A generalization for n integers was recently proposed in [14] and it is henceforth referred to as *3-term τJSF* because its average number of non-zero columns is equivalent to the 3-term JSF [14]. A 3-term τJSF has a probability of 0.5897 for a non-zero column [14] which yields a *Hamming weight*, i.e. the number of non-zero terms, $H(k) = 0.5897m$ on average. This paper considers the following *3-term multiple point multiplication*:

$$Q = k_1 P_1 + k_2 P_2 + k_3 P_3 \ . \tag{1}$$

2.3 Self-Certified Identity Based Signatures

In the following, a *self-certified identity based signature* scheme [3] based on Nyberg-Rueppel signatures [15] is outlined for groups over elliptic curves as presented by Brumley in [8].

First, an elliptic curve E and a base point G with prime order r is chosen and the Trusted Third Party (TTP) generates a domain private key s_D and computes and publishes a domain public key $W_D = s_D G$. Then, the TTP generates a private key for Alice's identity ID_A by calculating

$$\begin{aligned} (r_A, b_A) &= \textsc{compress}(uG) + \textsc{hash}(ID_A) \\ s_A &= u - s_D r_A \pmod{r} \end{aligned}$$

where u is an integer selected at random from the interval $[1, r-1]$ and COMPRESS compresses a point (x, y) to $(x, b(y))$ which requires only $m+1$ bits. HASH is a hash function.

Alice generates a signature (c, d) for a message $\mathcal{M}$ by calculating

$$\begin{aligned} c &= [vG]_x + \textsc{hash}(\mathcal{M}) \\ d &= v - s_A c \pmod{r} \end{aligned} \tag{2}$$

where v is a random integer such that $v \in [1, r-1]$ and $[vG]_x$ is the x-coordinate of vG.

Bob verifies the signature on the message $\mathcal{M}$ by first extracting Alice's public key W_A from (r_A, b_A) which are public by computing

$$W_A = \textsc{decompress}(r_A - \textsc{hash}(ID_A), b_A) - r_A W_D \tag{3}$$

where DECOMPRESS is the inverse operation of COMPRESS. Thus, (3) requires one point multiplication. After extraction, the validity of the signature is verified by checking

$$\textsc{hash}(\mathcal{M}) = c - [dG + cW_A]_x \pmod{r} \tag{4}$$

which requires two point multiplications. Verification and extraction can be simplified into the following 3-term multiple-point multiplication as shown in [8]:

$$dG + c(uG) - cr_A W_D$$

which obviously has the form of (1).

As signings, i.e. computations of (2), are computationally cheaper than verifications and they can be accelerated further with methods such as fixed-base windowing (see [16], for example), the performance of the scheme is bounded by verifications. 3-term multiple point multiplication dominates in the computational requirements of verification because decompression and subtraction are fast to compute. The hash can be computed simultaneously with point multiplication, and many fast and compact hash modules have been presented in the literature. Thus, the remainder of the paper focuses in accelerating (1).

3 Algorithms

This section introduces the algorithms which are used in computing (1). Point multiplications are computed using known algorithms which are reviewed in Sec. 3.1 but new algorithms are derived for precomputations in Sec. 3.2.

3.1 Elliptic Curve Point Multiplication

When two points on E_K are represented in *affine coordinates*, $\mathcal{A}$ for short, as (x_1, y_1) and (x_2, y_2), a point addition $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$ is given with the following formulae:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \tag{5a}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \tag{5b}$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \ . \tag{5c}$$

They have the cost $I + 2M + S + 8A$ where I, M, S and A denote the costs of inversion, multiplication, squaring and addition in $\mathbb{F}_{2^m}$, respectively. A negation of the point (x_1, y_1) is given by $(x_1, x_1 + y_1)$ and it has the cost of A. [16]

Because inversions are expensive, it is commonly preferred to represent points with three coordinates as (X, Y, Z) because then the number of inversions in point multiplication can be reduced to one. Coordinate system called *López-Dahab coordinates* [17], or $\mathcal{LD}$ for short, is used in this paper and a point (X, Y, Z) in $\mathcal{LD}$ represents the point $(X/Z, Y/Z^2)$ in $\mathcal{A}$ [17]. When points are represented in $\mathcal{LD}$, point addition $P_3 = P_1 + P_2$ can be computed as presented in [18] so that P_1 is in $\mathcal{LD}$ and P_2 in $\mathcal{A}$. This is referred to as the *mixed coordinate point addition* and it has the cost of only $8M + 5S + 8A$ on the NIST curve K-163. Frobenius map is (X^2, Y^2, Z^2) in $\mathcal{LD}$ and it is obviously cheap to compute. The $\mathcal{A} \mapsto \mathcal{LD}$ mapping is performed at the beginning simply as $(x, y, 1)$ but the $\mathcal{LD} \mapsto \mathcal{A}$ mapping requires $I + 2M + S$. However, as shown in (4), the y-coordinate is not needed in verification and, hence, the cost reduces to only $I + M$.

Finite fields $\mathbb{F}_{2^m}$ are typically represented with *polynomial basis* or *normal basis*. In polynomial basis, the field is constructed by using an irreducible polynomial with a degree of m. In normal basis, the set $\{\alpha, \alpha^2, \dots, \alpha^{2^{m-1}}\}$, where

α^{2^i} are linearly independent, is used as a basis and an element is represented as $a = \sum_{i=0}^{m-1} a_i \alpha^{2^i}$ where $a_i \in \{0, 1\}$. Multiplication is considered more efficient in polynomial basis. However, in normal basis squaring is simply a rotation of the bit vector and Frobenius maps are thus very cheap to compute. For this reason normal basis was chosen. Addition is computed with a simple bitwise exclusive-or (XOR). Inversion is computed with *Itoh-Tsujii inversion* [19] requiring exactly $(\lfloor \log_2(m-1) \rfloor + H(m-1) - 1)M + (m-1)S$ where $H(m-1)$ is the Hamming weight of $m-1$ [19]. As $m = 163$, the cost is $I = 9M + 162S$. Because squarings are cheap, multiplications dominate in I.

To summarize, the implementation computes (1) on the NIST K-163 (normal basis) with the binary method using a 3-term τJSF. Point additions are computed in mixed coordinates and, in the end, the x-coordinate is mapped to $\mathcal{A}$ by computing X/Z, where the inversion is computed with an Itoh-Tsujii inversion.

3.2 Precomputation

When (1) is computed with multiple point multiplication techniques, certain points need to be precomputed. These precomputations cannot be computed offline similarly as, e.g., in fixed-base windowing methods because points P_i are not fixed. Thus, precomputations are on the critical path and it is essential to compute them as fast as possible. In order to be able to use fast mixed coordinate point additions, precomputed points should be in $\mathcal{A}$. The first step in improving precomputations is to utilize the fact that the same inversion is computed in both $P_1 + P_2$ and $P_1 - P_2$ computations. The same fact has been previously used at least in [20] but it is shown in the following that it is also possible to save some additions.

Theorem 1 (Unified point addition and subtraction). *Given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on an elliptic curve E, $P_3^{(+)} = (x_3^{(+)}, y_3^{(+)}) = P_1 + P_2$ and $P_3^{(-)} = (x_3^{(-)}, y_3^{(-)}) = P_1 - P_2$ can be computed with the following formulae:*

$$\theta = (x_1 + x_2)^{-1} \tag{6a}$$
$$\lambda = (y_1 + y_2)\theta \quad \textit{and} \quad \lambda' = x_2\theta \tag{6b}$$
$$x_3^{(+)} = \lambda^2 + \lambda + x_1 + x_2 + a \tag{6c}$$
$$y_3^{(+)} = \lambda(x_1 + x_3^{(+)}) + x_3^{(+)} + y_1 \tag{6d}$$
$$x_3^{(-)} = x_3^{(+)} + \lambda'^2 + \lambda' \tag{6e}$$
$$y_3^{(-)} = (\lambda + \lambda')(x_1 + x_3^{(-)}) + x_3^{(-)} + y_1 \ . \tag{6f}$$

Proof. (6c) and (6d) are simply the point addition formulae (5b) and (5c), i.e. $(x_3^{(+)}, y_3^{(+)}) = P_1 + P_2$, and it remains to show that $(x_3^{(-)}, y_3^{(-)}) = P_1 - P_2$. Substituting (6c) into (6e) results in

$$\begin{aligned} x_3^{(-)} &= \lambda^2 + \lambda'^2 + \lambda + \lambda' + x_1 + x_2 + a \\ &= (\lambda + \lambda')^2 + (\lambda + \lambda') + x_1 + x_2 + a \end{aligned} \tag{7}$$

because $2\lambda\lambda' = 0$. As $-P_2 = (x_2, x_2 + y_2)$, (5a) yields

$$\lambda^{(-)} = \frac{y_1 + x_2 + y_2}{x_1 + x_2} = \frac{y_1 + y_2}{x_1 + x_2} + \frac{x_2}{x_1 + x_2} = \lambda + \lambda' \ . \tag{8}$$

Now substituting (8) into (6f) and (7) shows that $(x_3^{(-)}, y_3^{(-)}) = P_1 - P_2$. $\square$

Cost of computing (6a)–(6f) is only $I + 4M + 2S + 14A$. Thus, Theorem 1 saves $\mathsf{I} + 3\mathsf{A}$ compared (5a)–(5c). This is significant because inversion dominates in the cost of point addition.

Precomputations in 3-term (τ)JSF require 10 point additions or subtractions because points presented in Table 1 need to be available. Obviously, pairs (R_4, R_5), (R_6, R_7), (R_8, R_9), (R_{10}, R_{11}) and (R_{12}, R_{13}) can be computed using (6). Thus, the precomputations require only 5 unified point additions and subtractions. It should be noted that the use of unified point additions and subtractions does not restrict to JSF precomputations because similar pairs can be found, e.g., in precomputations involved in combings when integers are in NAF.

Table 1. Precomputed points and the corresponding columns in 3-term (τ)JSF

$k_3k_2k_1$	Point	$k_3k_2k_1$	Point	$k_3k_2k_1$	Point	$k_3k_2k_1$	Point
000	$R_0 = \mathcal{O}$	$10\bar{1}$	$R_7 = R_3 - R_1$	n/a		$\bar{1}01$	$-R_7$
001	$R_1 = P_1$	110	$R_8 = R_3 + R_2$	$00\bar{1}$	$-R_1$	$\bar{1}\bar{1}0$	$-R_8$
010	$R_2 = P_2$	$1\bar{1}0$	$R_9 = R_3 - R_2$	$0\bar{1}0$	$-R_2$	$\bar{1}10$	$-R_9$
100	$R_3 = P_3$	111	$R_{10} = R_8 + R_1$	$\bar{1}00$	$-R_3$	$\bar{1}\bar{1}\bar{1}$	$-R_{10}$
011	$R_4 = R_2 + R_1$	$11\bar{1}$	$R_{11} = R_8 - R_1$	$0\bar{1}\bar{1}$	$-R_4$	$\bar{1}\bar{1}1$	$-R_{11}$
$01\bar{1}$	$R_5 = R_2 - R_1$	$1\bar{1}1$	$R_{12} = R_9 + R_1$	$0\bar{1}1$	$-R_5$	$\bar{1}1\bar{1}$	$-R_{12}$
101	$R_6 = R_3 + R_1$	$1\bar{1}\bar{1}$	$R_{13} = R_9 - R_1$	$\bar{1}0\bar{1}$	$-R_6$	$\bar{1}11$	$-R_{13}$

Computational cost of precomputations can be reduced even further by using *Montgomery's trick* (see [16], for example) for computing the five inversions. Montgomery's trick is based on the observation that $1/\theta_1 = \theta_2(1/\theta_1\theta_2)$ and $1/\theta_2 = \theta_1(1/\theta_1\theta_2)$ and it operates as follows. Let $\theta_1, \theta_2, \ldots, \theta_n$ be the elements to be inverted. First, set $\gamma_1 = \theta_1$ and, for $i = 2, \ldots, n$, compute $\gamma_i = \gamma_{i-1}\theta_i$. Then invert γ_n^{-1} and compute $\theta_n^{-1} = \gamma_{n-1}\gamma_n^{-1}$. For $i = n-1, \ldots, 2$, compute $\gamma_i^{-1} = \theta_{i+1}\gamma_{i+1}^{-1}$ and $\theta_i^{-1} = \gamma_{i-1}\gamma_i^{-1}$. Finally, $\theta_1^{-1} = \theta_2\gamma_2^{-1}$. Montgomery's trick inverts n elements with the cost of $3(n-1)M + I$. [16]

However, Montgomery's trick is not directly applicable in 3-term JSF precomputations because it requires that all θ_i are known in advance. Let $R_i = (\hat{x}_i, \hat{y}_i)$ as defined in Table 1. The following inverses are needed in computing R_i: $\theta_1^{-1} = (\hat{x}_1 + \hat{x}_2)^{-1}$, $\theta_2^{-1} = (\hat{x}_1 + \hat{x}_3)^{-1}$, $\theta_3^{-1} = (\hat{x}_2 + \hat{x}_3)^{-1}$, $\theta_4^{-1} = (\hat{x}_8 + \hat{x}_1)^{-1}$ and $\theta_5^{-1} = (\hat{x}_9 + \hat{x}_1)^{-1}$ in which only $\hat{x}_1 = x_1$, $\hat{x}_2 = x_2$ and $\hat{x}_3 = x_3$ are known beforehand. In order to be able to use Montgomery's trick, $\hat{x}_8$ and $\hat{x}_9$ need to be presented by using x_1, y_1, x_2, y_2, x_3 and y_3.

Because $R_8 = (\hat{x}_8, \hat{y}_8) = P_3 + P_2$, it follows directly from (5a) and (5b) that

$$\theta_4^{-1} = \frac{(x_2 + x_3)^2}{(y_2 + y_3)^2 + (x_2 + x_3)(y_2 + y_3) + (x_2 + x_3)^2(x_1 + x_2 + x_3 + a)} \ . \quad (9)$$

Let θ_4' denote the denominator of (9). Similarly as above,

$$\theta_5^{-1} = \frac{(x_2 + x_3)^2}{\theta_4' + x_2 x_3} \quad (10)$$

and, again, let θ_5' denote the denominator of (10).

Now, Montgomery's trick can be used for computing inverses for the elements $\theta_1 = x_1 + x_2$, $\theta_2 = x_1 + x_2$, $\theta_3 = x_2 + x_3$, θ_4' and θ_5'. In order to get θ_4^{-1} and θ_5^{-1}, $\theta_4'^{-1}$ and $\theta_5'^{-1}$ are multiplied with $\theta_3^2 = (x_2 + x_3)^2$ as shown in (9) and (10). Finally, R_i can be computed with (6b)–(6f), i.e. by skipping the inversion of (6a). An algorithm is presented in Alg. 1.

Algorithm 1 Precomputation in 3-term (τ)JSF

Input: $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_3 = (x_3, y_3)$
Output: Precomputed points R_i as described in Table 1
$\theta_1 \leftarrow x_1 + x_2$; $\theta_2 \leftarrow x_1 + x_3$; $\theta_3 \leftarrow x_2 + x_3$
$\theta_4 \leftarrow (y_2 + y_3)^2 + \theta_3 \times (y_2 + y_3) + \theta_3^2 \times (x_1 + \theta_3 + a)$
$\theta_5 \leftarrow \theta_4 + x_2 \times x_3$
Compute inverses θ_i^{-1} with Montgomery's trick
$\theta_4^{-1} \leftarrow \theta_3^2 \times \theta_4^{-1}$; $\theta_5^{-1} \leftarrow \theta_3^2 \times \theta_5^{-1}$
$R_1 \leftarrow P_1$; $R_2 \leftarrow P_2$; $R_3 \leftarrow P_3$;
$R_{4,5} \leftarrow R_2 \pm R_1$; $R_{6,7} \leftarrow R_3 \pm R_1$; $R_{8,9} \leftarrow R_3 \pm R_2$
$R_{10,11} \leftarrow R_8 \pm R_1$; $R_{12,13} \leftarrow R_9 \pm R_1$

Table 2 lists the costs of 3-term (τ)JSF precomputations with the three techniques considered above, i.e. with 10 point additions (naïve) or 5 unified point additions and subtractions without (unified) or with (unified + Montgomery) Montgomery's trick. The methods presented above reduce the number of multiplications required in precomputations by 58 % in the case of $\mathbb{F}_{2^{163}}$ and Itoh-Tsujii inversion.

4 Implementation

This section presents the design in detail. The design is implemented on an Altera Stratix II EP2S180 DSP development board, professional edition [21], which includes an Altera Stratix II EP2S180F1020C3 FPGA [22].

The goal of the implementation is in maximizing the number of operations per second (ops) rather than in minimizing computation time of a single operation. The implementation is designed to be modular so that it can be easily parallelized

Table 2. Costs of 3-term (τ)JSF precomputations with different techiques. The values in the rightmost column present the number of multiplications when using an elliptic curve over $\mathbb{F}_{2^{163}}$ and Itoh-Tsujii inversion.

Method	Cost	$\mathsf{I} = 9\mathsf{M}$
Naïve	$10\,(\mathsf{I} + 2\mathsf{M} + \mathsf{S} + 8\mathsf{A}) + 5\mathsf{A}$	110M
Unified	$5\,(\mathsf{I} + 4\mathsf{M} + 2\mathsf{S} + 14\mathsf{A})$	65M
Unified + Montgomery	$\mathsf{I} + 17\mathsf{M} + 2\mathsf{S} + 9\mathsf{A} + 5\,(4\mathsf{M} + 2\mathsf{S} + 14\mathsf{A})$	46M

in order to increase ops. It consists of two main modules; namely, *converters* for finding 3-term τJSF for integers and *field arithmetic processors* (FAPs) with control logic for computing point multiplications. These modules are considered in Secs. 4.1 and 4.2, respectively.

There are certain parameters which define the performance and area requirements of an implementation. It is not obvious how these design parameters should be chosen and, thus, parameter space exploration is performed in Sec. 4.3 in order to find optimal parameters.

It should be noticed that, while side-channel attacks are a serious threat for many security applications in FPGAs [23], they are insignificant in this case because all information is public anyhow.

4.1 τNAF and 3-term τJSF Conversions

As mentioned in Sec. 2.2, integer k needs to be converted into a τ-adic expansion before point multiplication. Conversions to τNAF are performed as presented by the authors in [24]. Because three conversions are required in 3-term multiple point multiplication, there are basically two alternatives: either required conversions are computed with one τNAF converter resulting in a critical path of three conversions or with three τNAF converters and a critical path of one conversion. The latter alternative was chosen mainly for two reasons:

1. Latency is shorter, and
2. no storage for converted values is needed before τJSF conversions.

Once the integers are converted into τNAF, a 3-term τJSF is build up as presented in [14]. The algorithm of [14] was implemented so that the four most recent signed bits from the τNAF converters, which output their results in serial, are stored into three shift registers, each of which contains 4 signed bits. The values of the shift registers are input into a circuit that determines whether the values of all three registers are reducible or not. If they are reducible and there are no all-zero columns, then the values of the registers are updated with reduced values. In the 3-term case the value 1001 of a shift register is replaced by $00\bar{1}\bar{1}$, $\bar{1}00\bar{1}$ by 0011, 1010 by $01\bar{1}0$, and $\bar{1}0\bar{1}0$ by $0\bar{1}10$.

4.2 Point Multiplication

Point multiplication is computed with an architecture comprising an FAP and logic controlling it.

Field arithmetic processor. The FAP consists of adder, squarer, multiplier, storage RAM and instruction decoder.

The adder computes a bitwise XOR of two m-bit operands, and it has a latency of one clock cycle, i.e. $A = 1$. The squarer supports computation of multiple successive squarings, i.e. x^{2^d} where $x \in \mathbb{F}_{2^m}$ and d is an integer in the interval $[0, d_{\max}]$ with $d_{\max} = 2^5 - 1$. In normal basis, squaring is a rotation of the bit vector, and the squarer is a shifter which computes x^{2^d} in one clock cycle. The cost of d squarings is $S_d = 1$. This has serious implications because d successive Frobenius maps in $\mathcal{LD}$ can be computed with the cost of only $3S_d$ if $d \leq 31$ and Itoh-Tsujii inversion costs only $9M + 14S_d$ instead of $9M + 162S$.

Field multiplication is critical for the overall performance. Multiplication in normal basis is performed with a multiplier which is a digit-serial implementation of the *Massey-Omura multiplier* [25]. In a bit-serial Massey-Omura multiplier, one bit of the output is calculated in one clock cycle and, hence, m cycles are required in total. One bit z_i of the result $z = x \times y$ where $x, y, z \in \mathbb{F}_{2^m}$ is computed from x and y by using an F-function. The F-function is field specific, and the same F is used for all output bits z_i as follows: $z_i = F(x_{\lll i}, y_{\lll i})$, where $\lll i$ denotes cyclical left shift by i bits. Hence, a bit-serial implementation of the Massey-Omura multiplier requires three m-bit shift registers and one F-block. A bit-parallel implementation, where all bits z_i are computed in parallel, requires m F-blocks and an m-bit register for storing the result. [4, 25]

In practice, the bit-serial implementation requiring at least $m+1$ clock cycles is too slow and the bit-parallel implementation requires too much area. A good tradeoff is a digit-serial multiplier, where v bits are computed in parallel with v F-blocks. The F-block forms the critical path of an FAP and determines the maximum clock frequency. Thus, the maximum clock frequency can be increased by pipelining the F-blocks. As one clock cycle is required in loading the operands into the shift registers and each pipeline stage increases latency by one clock cycle, the latency becomes

$$M = \left\lceil \frac{m}{v} \right\rceil + c + 1 \tag{11}$$

where c is the number of pipeline stages inside the F-blocks, i.e. $c \geq 0$. In this paper, $c = 1$. It follows directly from (11) that, when $m = 163$, the number of F-blocks, v, should be chosen from the following set of integers:

$$\mathcal{F} : \{1 - 15, 17, 19, 21, 24, 28, 33, 41, 55, 82, 163\} \ .$$

All other values only increase area without decreasing latency.

The *storage RAM* is used for storing elements of $\mathbb{F}_{2^m}$. Stratix II devices include M512, M4K and M-RAM memory blocks and they contain 575, 4,608, and 589,824 bits of RAM, respectively [22]. Using embedded memory blocks is advantageous because more logic resources are saved for the actual computation. The storage RAM is implemented with M4Ks as a dual-port RAM and it is capable of storing W elements. A logical choice is $W = 256$ because, while in true dual-port mode, the widest mode that an M4K block can be configured to is 256×18-bits [22]. Thus, the storage RAM requires $\lceil 163/18 \rceil = 10$ M4Ks resulting in a storage capacity of 256×163-bits. This much storage space is rarely needed but it can be used for example for storing precomputed points. Moreover, selecting a smaller depth than 256 would not reduce the number of required M4Ks. Both writing and reading to and from the storage RAM require one clock cycle. However, the dual-port RAM can be configured into the read-during-write mode [22] which saves certain clock cycles as will be discussed in the following.

Control logic. The logic controlling the FAP consists of finite state machine (FSM) and ROM containing instruction sequences.

The instruction sequences are carefully hand-optimized and certain tricks are used in order to minimize latencies of point operations. The read-during-write mode can be used for reducing latencies. In order to maximize the advantages in this case, operations are ordered so that the result of the previous operation is used as an operand for the next operation whenever possible. This saves one clock cycle because the operands of the next operation can be read simultaneously while the result of the previous operation is being written.

Latency of computing $k_1P_1 + k_2P_2 + k_3P_3$ with a 3-term τJSF becomes

$$\underbrace{46M + 306}_{\text{Precomputation}} + \underbrace{(H(k) - 1)(8M + 47)}_{\text{Point additions and Frobenius maps}} + \underbrace{10M + 68}_{X/Z \text{ and interfacing}} \tag{12}$$

clock cycles where $H(k)$ is the number of non-zero columns in the 3-term τJSF.

Fig. 1 presents an example operation schedule of an implementation with one converter and two FAPs. The implementation computes five point multiplications in the example so that when the first integers and points arrive (data #1), it immediately starts computing a 3-term τJSF for the integers in the converter and precomputed points in the first FAP. Because a precomputation requires more time than a conversion, the computation time only consists of precomputation time and point multiplication time if there are resources available immediately at the arrival of data. This is the case for datas #1, #2 and #5. However, when data #4 arrives, conversion can be started instantaneously but precomputation can be started when the second FAP becomes available. The situation is even worse for data #3 because, when it arrives, there are no converters or FAPs available, and thus even longer delay occurs.

4.3 Parameter Exploration

Free parameters in the design are the numbers of F-blocks, v, and the number of parallel FAPs, p, of which only $v \in \mathcal{F}$ determines the latency of a single point

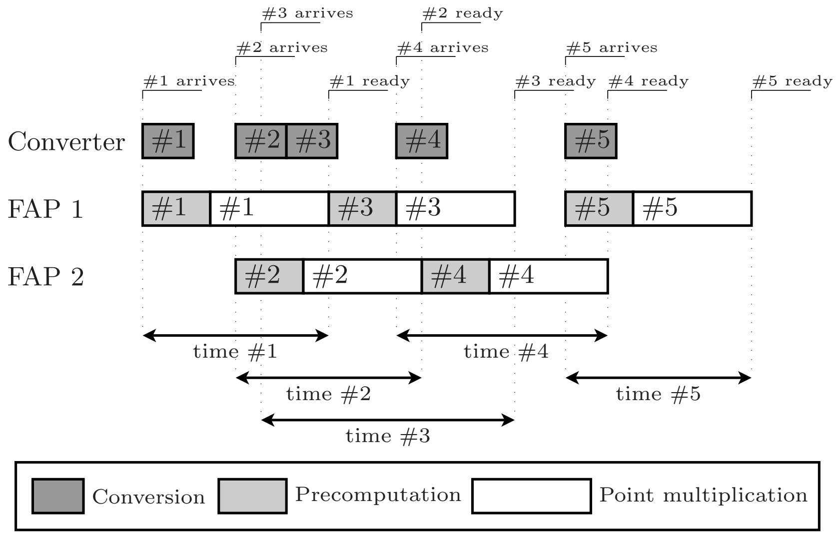

Fig. 1. Computation schedule example in the case of two FAPs and one converter

multiplication, and p only increases ops. If the objective is in minimizing computation time, it is obvious how the parameters effect to the result. That is, when v grows, computation time decreases. However, if the objective is in maximizing ops with parallel FAPs, the situation is more complicated. The reason for this is that, when an area constraint is set, v determines the maximum number of FAPs, $p_{\max}$, that can be included under the constraint. That is, the larger v is the smaller is $p_{\max}$. Thus, there is a need for an analysis on v and $p_{\max}$ setups.

Estimates of area consumption are needed in order to analyze v and p setups. These estimates were received by synthesizing an FAP with Quartus II 6.0 SP1 design software, and an approximation of the area is given by

$$A(p, v) = p(A_c + vA_F)$$

where $A_F = 147\,\text{ALMs}$ (Adaptive Logic Modules) is the size of an F-block and $A_c = 1202\,\text{ALMs}$ is the size of other blocks in the FAP, i.e. adder, squarer, control logic, etc.

Field multiplication determines point multiplication latency together with $H(k)$ as shown in (12). It is assumed in the following analysis that JSFs have an average number of non-zero columns, and such JSFs are henceforth referred to as *average JSFs*. Thus, latency depends only on the latency of field multiplication. The critical path determining the maximum clock frequency does not depend on v and, thus, it is assumed that all FAPs operate at the same clock frequency; see Sec. 4.2. Based on the results obtained from Quartus II, it is assumed that the clock frequency is 160 MHz.

Fig. 2 plots point multiplication time and ops as functions of v (and $p_{\max}$) when an area constraint of 75 % of ALMs is given for FAPs. The remaining 25 %

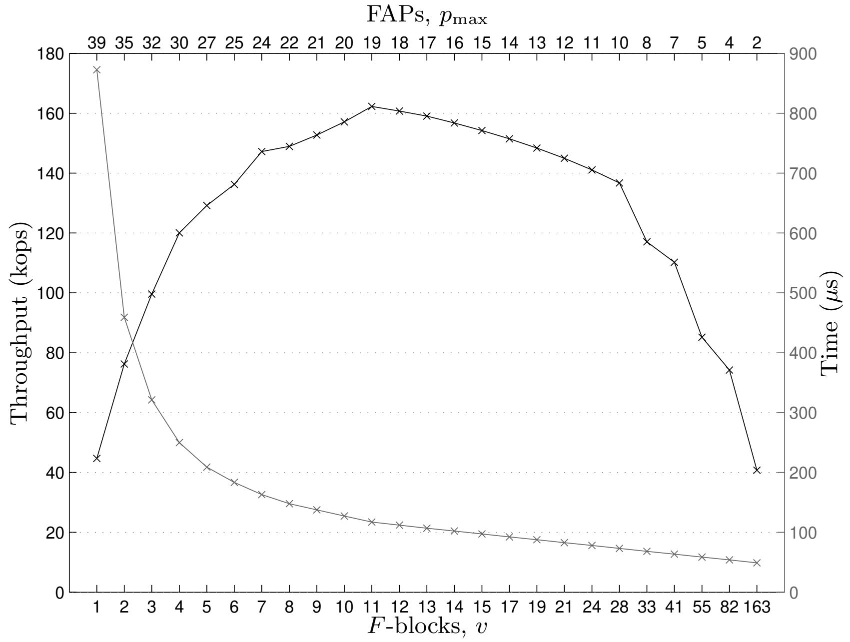

Fig. 2. Computation time and ops estimates with different v and $p_{\max}$ setups. The black line indicates throughput (ops) and its value is read from the vertical axis on the left. The grey line is the computation time and its value is in the vertical axis on the right. The number of F-blocks, v, determines the maximum number of FAPs, $p_{\max}$, which fit into the device under the given constraint of 75 % of ALMs, and $p_{\max}$ can be found on the upper horizontal axis.

are reserved for the converters, interfacing, etc. Stratix II S180 includes 71,760 ALMs in total [22]. The maximum ops is received when $v = 11$. In that case, an FAP can compute 3-term multiple point multiplication in 117 μs and 19 FAPs fit into Stratix II S180 resulting in the maximum throughput of 162,000 ops.

Fig. 2 leads to a conclusion that tolerating slightly longer computation latencies can lead to major increases in ops. Furthermore, it can be seen that $v < 11$ should be never selected because higher ops can be achieved with shorter computation time. However, all $v \geq 11$ with $v \in \mathcal{F}$ are justified. If $v > 11$ is selected resulting in shorter computation time, then one must tolerate fewer ops. The design implemented in this paper uses the setup $p = 19$ and $v = 11$ in order to maximize ops.

The number of converters must be selected so that they do not become a bottleneck. If only a few converters are implemented, the average end-to-end computation time grows because data needs to wait for free converters longer; see Sec. 4.2. However, if many converters are implemented, the area constraint for FAPs needs to be lowered resulting in a decrease in performance.

5 Results

The design presented in Sec. 4 was written in VHDL and synthesized for the Stratix II FPGA by using Quartus II 6.0 SP1. Simulations were performed with ModelSim SE 6.1b. The design comprising 4 τJSF converters, 19 FAPs ($v = 11$) and FIFO buffers separating blocks requires in total 67,467 ALMs which is 94 % of the device resources and 240 M512 (26 %) and 305 M4K (40 %) memory blocks. The converters and the FAPs are separated into different clock domains and they have the maximum clock frequencies of 82.38 MHz and 167.50 MHz, respectively.

A phase-locked loop (PLL) in Stratix II was used for creating 82 MHz and 164 MHz clocks for the converters and FAPs. The converters compute a τJSF for three integers on average in 499 clock cycles which equals to 6.9 μs. The average latency of 18,733 clock cycles for a 3-term multiple point multiplication including precomputations is given by (12) which equals to 114.2 μs. This is also the minimum time in which the implementation computes a 3-term multiple point multiplication with an average JSF because conversions and precomputations are computed in parallel. Theoretically, the implementation is capable of performing up to 166,000 verifications per second.

To the authors' knowledge, the fastest published FPGA implementation for the NIST curve K-163 was presented by Dimitrov et al. in [26] where a 1-term point multiplication requires 35.75 μs on Xilinx Virtex-II which would result in approximately 9,300 verifications per second. This was achieved by representing k with multiple-base expansions [26]. The implementation was optimized for low latency but, naturally, it could be parallelized in order to increase ops and rough estimates are given next. Because the FAP used in [26] requires 6,494 slices, a parallel implementation outperforming 166,000 verifications per second would need 18 FAPs resulting in approximately 117,000 slices without converters. Thus, the implementation would be too large to fit any FPGA available at the moment. However, the FAP with $v = 24$ [26] is probably larger than the one optimizing ops. Thus, the idea presented in this paper could be used, most probably resulting in more ops with fewer resources.

The results have shown that using parallel FAPs and 3-term τJSF enables considerable performance increases and the implementation presented here outperforms all previously published implementations if ops are considered.

6 Conclusions

This paper presented an efficient implementation designed specifically for rapid verification of self-certified identity based signatures. It was shown that it is possible to compute up to 166,000 verifications per second with a single Altera Stratix II FPGA. The results have significance in many cryptosystems whose performance is bounded by demanding signature verifications. One example is PLA where packets are verified by using cryptographic signatures.

The high performance was achieved by using parallel processors which were carefully optimized. Instead of concentrating in minimizing computation time of a single processor, the objective was shifted to maximizing the number of verifications per second computed by the parallel processors. It was concluded that major increases in ops can be achieved by tolerating slightly longer computation times, i.e. by using multiple smaller processors instead of only a few large processors. The idea can be easily generalized to other elliptic curve cryptosystems and implementation platforms.

Future work. Because field multiplication dominates in the performance and area requirements, it is of interest to optimize the multiplier architecture. One possibility is to use polynomial basis instead of normal basis. Polynomial bases are commonly preferred in implementing elliptic curve cryptosystems in hardware and they could offer some performance improvements. Another option is to use a more efficient architecture for normal basis multiplication.

A counterpart implementation which produces self-certified signatures will be designed. As mentioned, high performance is easier to achieve in signing because fewer point multiplications are needed and it is possible to use such methods as fixed-base windowing. Thus, performance should not be a problem. However, countermeasures against side-channel attacks are needed in signing acceleration in order to ensure confidentiality of private keys.

Although point multiplications are the most expensive operations in signing and verification, also other operations, such as hash functions, are needed and they will be included into the implementations in the future.

Acknowledgments. The authors thank Billy Brumley from the Laboratory for Theoretical Computer Science at TKK for many valuable discussions. The authors also express their gratitude to the anonymous reviewers who gave a number of excellent comments and improvement suggestions.

References

1. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation **48** (1987) 203–209
2. Miller, V.: Use of elliptic curves in cryptography. In: Advances in Cryptology — CRYPTO '85. LNCS 218, Springer (1985) 417–426
3. Ateniese, G., de Medeiros, B.: A provably secure Nyberg-Rueppel signature variant with applications. Cryptology ePrint Archive, Report 2004/093 (2004)
4. National Institute of Standards and Technology (NIST): Digital signature standard (DSS). Federal Information Processing Standard (2000) FIPS PUB 186-2.
5. Koblitz, N.: CM-curves with good cryptographic properties. In: Advances in Cryptology — CRYPTO '91. LNCS 576, Springer (1991) 279–287
6. Candolin, C., Lundberg, J., Kari, H.: Packet level authentication in military networks. In: Proceedings of the 6th Australian Information Warfare & IT Security Conference. (2005)
7. Candolin, C.: Securing military decision making in a network-centric environment. PhD thesis, Helsinki University of Technology (2005)

8. Brumley, B.B.: Efficient three-term simultaneous elliptic scalar multiplication with applications. In: Proceedings of the 11th Nordic Workshop on Secure IT Systems, NordSec 2006. (2006) 105–116
9. Solinas, J.A.: Efficient arithmetic on Koblitz curves. Designs, Codes and Cryptography **19**(2–3) (2000) 195–249
10. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4) (1985) 469–472
11. Solinas, J.A.: Low-weight binary representations for pairs of integers. Technical Report CORR 2001-41, University of Waterloo, Centre for Applied Cryptographic Research (2001)
12. Proos, J.: Joint sparse forms and generating zero columns when combing. Technical Report CORR 2003-23, University of Waterloo, Centre for Applied Cryptographic Research (2003)
13. Ciet, M., Lange, T., Sica, F., Quisquater, J.J.: Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms. In: Advances in Cryptology — EUROCRYPT 2003. LNCS 2656, Springer (2003) 388–400
14. Brumley, B.B.: Left-to-right signed-bit τ-adic representations of n integers. In: Proceedings of the 8th International Conference on Information and Communications Security, ICICS 2006. LNCS 4307, Springer (2006) 469–478
15. Nyberg, K., Rueppel, R.A.: A new signature scheme based on the DSA giving message recovery. In: Proceedings of the 1st ACM conference on Computer and Communications Security, ACM Press (1993) 58–61
16. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer (2004)
17. López, J., Dahab, R.: Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. In: Selected Areas in Cryptography, SAC'98. LNCS 1556, Springer (1998) 201–212
18. Al-Daoud, E., Mahmod, R., Rushdan, M., Kilicman, A.: A new addition formula for elliptic curves over $GF(2^n)$. IEEE Transactions in Computers **51**(8) (2002) 972–975
19. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. Information and Computation **78**(3) (1988) 171–177
20. Okeya, K., Takagi, T., Vuillaume, C.: Efficient representations on Koblitz curves with resistance to side channel attacks. In: Proceedings of the 10th Australasian Conference on Information Security and Privacy, ACISP 2005. LNCS 3574, Springer (2005) 218–229
21. Altera: Stratix II EP2S180 DSP Development Board, Reference Manual. (2005)
22. Altera: Stratix II Device Handbook. (2006)
23. Standaert, F.X., Peeters, E., Rouvroy, G., Quisquater, J.J.: An overview of power analysis attacks against field programmable gate arrays. Proceedings of the IEEE **94**(2) (2006) 383–394
24. Järvinen, K., Forsten, J., Skyttä, J.: Efficient circuitry for computing τ-adic non-adjacent form. In: Proceedings of the IEEE International Conference on Electronics, Circuits and Systems, ICECS 2006, IEEE (2006) 232–235
25. Wang, C.C., Troung, T.K., Shao, H.M., Deutsch, L.J., Omura, J.K., Reed, I.S.: VLSI architectures for computing multiplications and inverses in $GF(2^m)$. IEEE Transactions in Computers **34**(8) (1985) 709–717
26. Dimitrov, V.S., Järvinen, K.U., Jacobson, M.J., Chan, W.F., Huang, Z.: FPGA implementation of point multiplication on Koblitz curves using Kleinian integers. In: Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006. LNCS 4249, Springer (2006) 445–459