Implementation of PSEC-KEM (secp256r1 and secp256k1) on Hardware and Software Platforms

Final Project Report

Sujoy Sinha Roy and Debdeep Mukhopadhyay
Indian Institute of Technology Kharagpur
West Bengal

January 2012

Contents

1 Introduction

PSEC-KEM is a provably secure key encapsulation mechanism. It is used to realize key agreement schemes. This document describes the software implementing of PSEC-KEM for elliptic curves with efficiently computable endomorphisms.

1.1 Scope

The implementation is compatible with [2] document. The software implementation of PSEC-KEM is over the 256 bit prime field with parameters $secp256k1$.

1.2 Constraints and Assumptions

- We assume all elliptic curve points are in uncompressed format.
- The software implementations of PSEC-KEM are targeted for testing functional correctness of the future hardware implementation on FPGAs.
- The software implementation is on Linux operating system and uses GMP library [7] for manipulating large integers.
- The random number used during encryption is generated using *mpz_urandomm* function available in GMP library.
- Open source code is used for the hash function. The source codes for the hash function is obtained from [6].

1.3 Glossary

PSEC-KEM	Provably Secure Elliptic Curve Key Encapsulation Mechanism
GF	Galois Field
ECCP	Elliptic Curve Crypto Processor
KDF	Key Distribution Function
FPGA	Field Programmable Gate Array

1.4 References

[1] NTT Information Sharing Platform Laboratories, NTT Corporation. *Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters (Version 2.0), Working Draft (January 27, 2010).*

[2] NTT Information Sharing Platform Laboratories, NTT Corporation. *PSEC-KEM Specification (Version 2.0).* June 2007.

[3] NTT Information Sharing Platform Laboratories, NTT Corporation. *Standards for Efficient Cryptography, SEC X.1: Supplemental Document for Odd Characteristic Extension Fields, Working Draft (Version 0.7).* May 2009.

[4] NTT Information Sharing Platform Laboratories, NTT Corporation. *Standards for Efficient Cryptography, SEC X.2: Recommended Elliptic Curve Domain Parameters, Working Draft (Version 0.6).* August 2008.

[5] Certicom Research. *Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography (Version 1.0)*. September 2000.

[6] SHA Opencores, *http://opencores.org/project,sha_core*

[7] The GNU Multiple Precision Arithmetic Library, *http://gmplib.org/*.

[8] D. Hankerson, A. Menezes, S. Vanstone, "Guide to Elliptic Curve Cryptography".

[9] R. P. Gallant, R. J. Lambert, S. A. Vanstone, "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms", Crypto 2001.

[10] T. Guneysu and Christof Paar, "Ultra High Performance ECC over NIST Primes on Commercial FPGAs", CHES 2008

2 Functional Description of PSEC-KEM for 256 bit Primes

The encryption and decryption algorithms for ES-PSEC-KEM are specified in [2]. Here we present the algorithms for the 256 bit prime field with parameters $secp256k1$. As per document [2], we have taken the octet length parameters $pLen = 32$, $hLen = 32$ and $keyLen = 32$.

2.1 Encryption Operation

Input: Public key W (which is sP, where s is private key)

Output: Session key k, delivery string c_0.

Assumptions: Public key W is valid.

Steps:

1. Generate a random binary string $r \in \{0,1\}^{256}$.
2. Let $H := KDF(I2OSP(0,4)||r, pLen + 16 + keyLen)$.
3. Parse $H = t||k$, where octet length of t is $pLen + 16$; octet length of k is $keyLen$.
4. Let $\alpha := OS2FEP(t)\ mod\ p$.
5. Let $C_1 := \alpha P$.
6. Let $Q := \alpha W$.
7. Let $c_2 := r \bigoplus KDF(I2OSP(1,4)||ECP2OSP(C_1)||PECP2OSP(Q), hLen)$.
8. Let $c_0 := ECP2OSP(C_1)||c_2$.
9. Output (k, c_0).

Output k is the secret session key. The string c_0 is delivered to the receiver party.

2.2 Decryption Operation

In the receiver side, after receiving c_0, a decryption is done to generate the secret session key k. The decryption algorithm takes private key s.

Input: Private key s, received string c_0.
Output: Session key k
Assumptions: Public key W is valid.
Steps:

1. If the octet length of c_0 is less than or equal to $hLen$, assert INVALID and stop.
2. Parse $c_0 = g||c_2$, where g and c_2 are octet strings such that the octet length of c_2 is $hLen$.
3. Let $C_1 := OS2ECPP(g)$. If OS2ECPP asserts INVALID, assert INVALID and stop.
4. Let $Q := sC_1$.
5. Let $r := c_2 \bigoplus KDF(I2OSP(1,4)||ECP2OSP(C_1)||PECP2OSP(Q), hLen)$.
6. Let $h := KDF(I2OSP(0,4)||r, pLen + 16 + hLen)$.
7. Parse $h = t||k$, where the octet length of t is $pLen + 16$; the octet length of k is $keyLen$.
8. Let $\alpha := OS2IP(t)\ mod\ p$.
9. Check $C_1 := \alpha P$. If it holds output k. Otherwise, assert INVALID and stop.

3 Elliptic Curves with Efficiently Computable Endomorphisms

Let E be an elliptic curve defined over a field K. An endomorphism ϕ of E over K is a map $\phi : E \to E$ such that $\phi(\mathcal{O}) = \mathcal{O}$ and $\phi(P) = (g(P), h(P))$ for all points P on E. Here g and h are rational functions whose coefficients are in K. Let p be a large prime with $p \equiv 1\ (mod\ 3)$, and E be an elliptic curve defined over $GF(p)$ of the form

$$E : y^2 = x^3 + b$$

Let β be an order three element in $GF(p)$. Then the map $\phi : E \to E$ defined by $\phi(x, y) \to (\beta x, y)$ and $\phi(\mathcal{O}) = \mathcal{O}$ is an endomorphism of E over $GF(p)$. The characteristic polynomial of ϕ is $\phi^2 + \phi + 1 = 0$.

3.1 Integer Representation of an Endomorphism

Let the order of the elliptic curve group i.e. $\#E$ has the largest prime factor n such that n^2 does not divide $\#E$. Then the group E has exactly one subgroup of order n. Let the subgroup be generated be a base point P of order n. There exists an integer $\lambda \in [1, n-1]$ such that

$$\phi(P) = \lambda P$$

where P is not the point at infinity. The integer λ is the root of $\lambda^2 + \lambda + 1 \equiv 0\ (mod\ n)$.

3.2 Accelerating Point Multiplication using Endomorphism

Let the scalar k can be expressed as

$$k = k_1 + k_2\lambda\ (mod\ n) \tag{1}$$

where k_1 and k_2 has almost half the length of k. The scalar multiplication kP can be written as

$$\begin{aligned} kP &= k_1P + k_2\lambda P \\ &= k_1P + k_2\phi(P) \\ &= k_1P + k_2Q \end{aligned} \tag{2}$$

where $Q = \phi(P)$. Since computation of $\phi(P)$ is easy (as it just requires a field multiplication), the computation cost of the scalar multiplication can be practically reduced to half by using the simultaneous point multiplication algorithm [9].

3.3 Decomposition of Scalar

Decomposition technique can be found in Algorithm 3.74 of [8]. For secp256k1, the values of a_1, b_1, a_2 and b_2 are as follows.
$\beta = 55594575648329892869085402983802832744385952214688224221778511981742606582254$
$\lambda = 37718080363155996902926221483475020450927657555482586988616620542887997980018$
$a1 = 64502973549206556628585045361533709077$
$b1 = -303414439467246543595250775667605759171$
$a2 = 367917413016453100223835821029139468248$
$b2 = 64502973549206556628585045361533709077$

4 Design of PSEC-KEM

In this section, we describe software and hardware architectures for PSEC-KEM.

4.1 Software Implementation of PSEC-KEM

We have implemented both $secp256r1$ and $secp256k1$ using C for software platforms. Large integer numbers are handled using standard GMP library [7]. The implementations use standard SHA256 source code available at [6], to construct the key derivation function (KDF). Standard affine coordinate arithmetic is used [8] to perform scalar multiplication. Correctness of the implementations were checked using GP/PARI tool.

4.2 Finite Field Primitives on Hardware Platforms

PSEC-KEM involves elliptic curve scalar multiplication in prime field. Efficiency of scalar multiplier depends on the efficiency of finite field primitives. In this section, we describe implementations of different finite field primitives used in PSEC-KEM.

4.2.1 Adder and Subtracter

Adders perform addition of 256 bit integers and thus carry propagation is a big challenge in designing adders. If the adders are implemented using Look-Up-Tables (LUT) present in FPGAs, then large amount of delay is incurred. Modern FPGAs provide dedicated DSP adders to perform fast integer addition. We have checked the performance of PSEC-KEM for DSP and LUT based adders and subtracters.

4.2.2 Multiplier

Prime field multiplier is the most important finite field primitive for elliptic curve scalar multiplication. Each point doubling and point addition during scalar multiplication involves several field multiplications. For fast scalar multiplication, the multiplier should be fast. For efficient implementation of large integer multiplier, we split any $2l$ bit operand into two l bit operands.

$$a \cdot b = a_h \cdot b_h 2^{2l} + [a_h \cdot b_l + a_l \cdot b_h]2^l + a_l \cdot b_l$$

For multiplications of 256 bit integers, we keep $h = l = 128$ bits. Any 256 bit multiplication is performed in the following steps: Then the multiplication of two $2l$ bit operands a and b is performed in the following way

1. Perform two multiplications $a_h \cdot b_l$ and $a_l \cdot b_h$ using the two 128 bit multipliers. The multiplication results are stored in 256 bit registers R1 and R2 respectively.
2. After completion of the multiplications in Step 1, perform the multiplications $a_l \cdot b_l$ and $a_h \cdot b_h$. Compute the addition $a_h \cdot b_l + a_l \cdot b_h = R1 + R2$ using the adder in parallel with the multiplications. The result of addition is 257 bit and is stored in R3. After completion of the two multiplications, store the results in R1 and R2 respectively.
3. Construct 257 bit W1={R2[128:0],R1[255:128]} and 127 bit W2=R2[255:129]. Perform 257 bit addition R3+W1 and store the 258 bit result in R3.
4. Compute final addition R3[msb] + W2 and write the 127 bit result on R4. Finally the multiplication result is $a \cdot b = \{R4, R3[256:0], R1[127:0]\}$.

The architecture for 256 bit multiplier is shown in Figure 1. Similar strategy is applied for the 128 bit multiplications. When the operand size reduces to 64 bits, basic school-book integer multipliers are applied. For a single 256 bit multiplication, the multiplier takes 10 clock cycles to complete the multiplication.

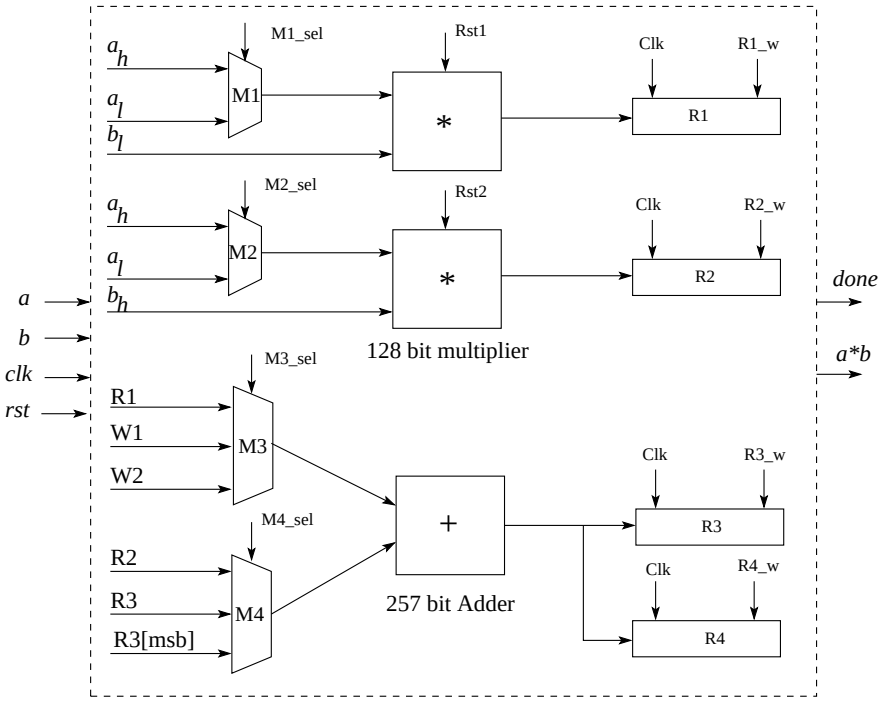

Figure 1: Implementation of 256 bit Integer Multiplier on FPGAs

4.2.3 Modular Reduction

Modular reduction is required for reducing an element in $GF(p)$. Due to special structure of the prime number in $secp256r1$, we have used fast modular reduction scheme [8]. The modular reduction scheme is shown below.

Fast Modular Reduction for $secp256r1$

Input: An integer $c = (c_{15}, \cdots, c_2, c_1, c_0)$ in base 2^{32} with $0 \leq c < p^2$.
Output: c mod p
Steps:

1. $s_1 = (c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$.
2. $s_2 = (c_{15}, c_{14}, c_{13}, c_{12}, c_{11}, 0, 0, 0)$.
3. $s_3 = (0, c_{15}, c_{14}, c_{13}, c_{12}, 0, 0, 0)$.
4. $s_4 = (c_{15}, c_{14}, 0, 0, 0, c_{10}, c_9, c_8)$.
5. $s_5 = (c_8, c_{13}, c_{15}, c_{14}, c_{13}, c_{11}, c_{10}, c_9)$.
6. $s_6 = (c_{10}, c_8, 0, 0, 0, c_{13}, c_{12}, c_{11})$.
7. $s_7 = (c_{11}, c_9, 0, 0, c_{15}, c_{14}, c_{13}, c_{12})$.
8. $s_8 = (c_{12}, 0, c_{10}, c_9, c_8, c_{15}, c_{14}, c_{13})$.
9. $s_9 = (c_{13}, 0, c_{11}, c_{10}, c_9, 0, c_{15}, c_{14})$.
10. Return $c = (s_1 + 2s_2 + 2s_3 + s_4 + s_5 - s_6 - s_7 - s_8 - s_9)$ mod p.

A single 260 bit adder cum subtracter is used to perform the operations. Finally, when c is positive and less than p, we get the reduced output. The modular reduction generally takes lesser number of clock cycles (8 to 10) compared to the 256 bit multiplier.

Since the 256 bit prime number used in $secp256k1$ is not a NIST recommended prime, to the best of our knowledge, fast modular reduction scheme for $secp256k1$ is not available in reported literature. To keep modular reduction fast for $secp256k1$, we have developed fast modular reduction technique for the prime moduli. The optimizations in the modular reduction technique are done with a target to keep clock cycle requirement minimum on hardware platforms. Similar optimizations may be achieved for software platforms. The following steps describe the modular reduction scheme for $secp256k1$.

Fast Modular Reduction for $secp256k1$

Input: A 512 bit integer a with $0 \leq a < p^2$.
Output: a mod p
Steps:

1. c0 = a[255:0];
2. c1 = a[511:256];
3. w0 = c0;
4. w1 = {c1[223:0], 32'd0};
5. w2 = {c1[246:0], 9'd0};
6. w3 = {c1[247:0], 8'd0};
7. w4 = {c1[248:0], 7'd0};
8. w5 = {c1[249:0], 6'd0};
9. w6 = {c1[251:0], 4'd0};
10. w7 = c1;

11. k1 = c1[255:252] + c1[255:250];
12. k2 = k1 + c1[255:249];
13. k3 = k2 + c1[255:248];
14. k4 = k3 + c1[255:247];
15. s1 = k4 + c1[255:224];
16. k11 = {s1, 2'd0} + {s1, 1'd0} + s1;
17. k12 = {k11, 7'd0};
18. k13 = {s1, 4'd0} + s1;
19. k14 = {s1, 6'd0} + k13;
20. k = {s1, 32'd0} + k12 + k14;
21. s = c0 + k + w1 + w2 + w3 + w4 + w5 + w6 + w7;
22. Return s mod p.

The fast reduction hardware, which follows the above steps takes 9-10 clock cycles to perform one modular reduction.

4.2.4 Finite Field Inversion

Finite field inversion is required during elliptic curve scalar multiplication. The finite field inverter used in this hardware implementation uses binary Euclidean algorithm (BEA) to compute inversion.

Binary Euclidean Inversion

Input: Prime p and $a \in [1, p-1]$.
Output: a^{-1} mod p.

1. $u \leftarrow a, v \leftarrow p$.
2. $x \leftarrow 1$, $y \leftarrow 0$.
3. While ($u \neq 1$ and $v \neq 1$) do
   - 3.1 While u is even do
     $u \leftarrow u/2$.
     If x is even then $x \leftarrow x/2$; else $x \leftarrow (x+p)/2$.
   - 3.2 While v is even do
     $v \leftarrow v/2$.
     If y is even then $y \leftarrow y/2$; else $y \leftarrow (y+p)/2$.
   - 3.3 If $u \geq v$ then: $u \leftarrow u - v$, $x \leftarrow x - y$ mod p;
     Else: $v \leftarrow v - u$, $y \leftarrow y - x$ mod p.
4. If $u = 1$ then return x; else return y.

The architecture for BEA hardware is shown in Figure 2. The hardware for BEA uses four 256 bit registers u, v, x and y. The register contents are updated as per the above algorithm. In Figure 2, hardware for updating u and x registers are shown. Similar hardware are used to update v (same as u) and y (same as x) registers. Every register has its write enable signal. In Figure 2, signal *u_write* and *x_write* are used for enabling writing on u and x registers. Control signals are generated to drive a finite state machine for computing inverse. Computation in step 3.3 of BEA requires comparison between u and v. In the BEA hardware, there are two dedicated 256 bit subtracters for computing $u - v$ and $v - u$. In Figure 2, Sub1 computes $u - v$. Comparison between u and v are performed by checking the output carry bits. In step 3.3, if value of x or y becomes negative, then the prime moduli is added to the negative value to make the value positive. Finally the inverse is obtained from the multiplexer M9 depending on u and v. Average clock cycle requirement for computing inverse is around 850.

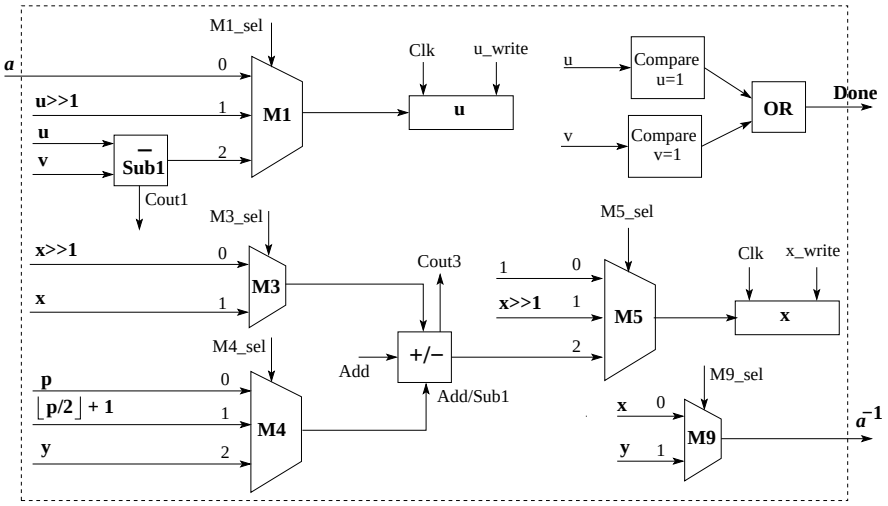

Figure 2: Hardware for Computing Inversion using BEA

4.3 Hardware Architecture for the Elliptic Curve Cryptoprocessor

In this section we present hardware architecture for elliptic curve scalar multipliers on both $secp256r1$ and $secp256k1$.

4.3.1 Hardware Architecture for ECCP in secp256r1

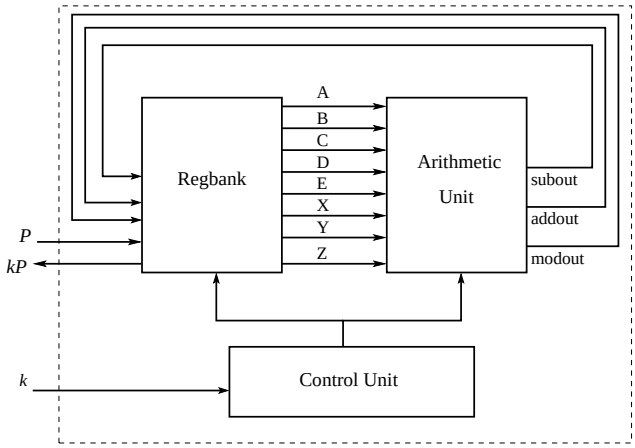

Figure 3: Block Diagram of Elliptic Curve Cryptoprocessor for $secp256r1$

Here we present an overview of the hardware architecture of the scalar multiplier on 256 bit prime field with specifications $secp256r1$ [1]. The elliptic curve crypto processor (ECCP) (Figure 3) takes as input a scalar k and produces the product kP, where $P = (Px, Py)$ is a base point on the curve. Since the second scalar multiplications in PSEC-KEM during encryption involves a point which is not fixed, the point is an input to the ECCP. Scalar multiplication algorithm uses standard double and add formulae and starts from the MSB side of the scalar. Point doublings are performed in Jacobian projective coordinate system, while point additions are done in affine-Jacobian coordinate system [8]. We have found that for left-to-right scalar multiplications on prime fields, the above choice of coordinate systems are most efficient in reducing overall number of squaring and multiplication operations. Finally, at the end of point addition and doubling series, one field inversion is performed for converting projective coordinate result to affine coordinate. Arithmetic operations during scalar multiplication involves field additions, squarings, and multiplications. These finite field operations are performed in the arithmetic unit (Figure 4). Squaring operations are performed as

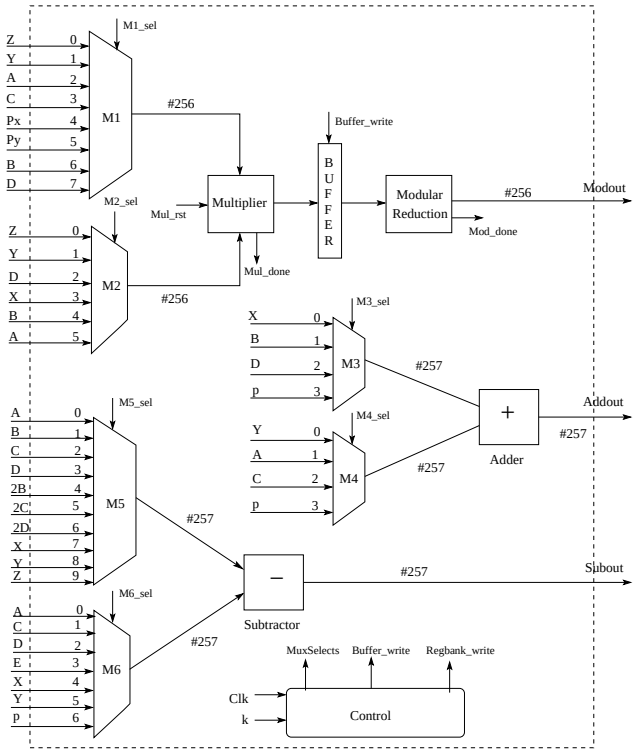

Figure 4: Arithmetic Unit for $secp256r1$

multiplications. Thus an elliptic curve point addition requires 11 multiplications, while doubling requires 8 multiplications. Since the result of multiplication is 512 bits, modular reduction is performed to reduce it to $GF(p)$ element. To increase efficiency of the scalar multiplier, we keep the multiplier and the modular reduction units in pipeline. Since there are several field multiplications during point addition or doubling, the order of different multiplications are arranged in such a way that in most of the consecutive multiplications are independent of each other. In such cases, while modular reduction is performed over a multiplication result, the multiplier can be used to perform the next independent multiplication. Such optimizations result in lesser number of clock cycle consumption for point doubling and addition. Intermediate results are stored in registers present in Regbank (Figure 3). The registers present in Regbank are A, B, C, D, E, X, Y and Z and have size either 256 bits or 257 bits. All registers are implemented using flip-flops present in FPGA slices. A control unit generates control signals in every clock cycle and drives the finite state machine for scalar multiplication. Using projective coordinates has the overhead that the result has to be converted from projective to affine coordinates. This requires an inverse to be computed followed by two multiplications. The inverse is computed BEA block. A (point doubling + point addition) for nonzero key bit takes 210 clock cycles, while a single point double for zero key bit takes 91 clock cycles. Overall clock cycle requirement for point addition and doublings for the 256 bit random scalar is around $38,500$.

The point doubling and addition steps for the scalar multiplier are shown below. Equations that appear in the same line are performed in parallel.

Point Doubling Equations for secp256r1 :

1. $U \leftarrow Z^2$;
2. $A \leftarrow U \ (mod\ p)$; $U \leftarrow Y^2$; $D \leftarrow X + A \ (mod\ p)$;
   $C \leftarrow 2D \ (mod\ p)$; $A \leftarrow X - A \ (mod\ p)$;
   $D \leftarrow D + C \ (mod\ p)$;
3. $B \leftarrow U \ (mod\ p)$; $U \leftarrow A \cdot D$; $B \leftarrow 2B \ (mod\ p)$;
   $C \leftarrow 2B \ (mod\ p)$;

4. $A \leftarrow U \ (mod\ p)$; $U \leftarrow C \cdot X$;
5. $D \leftarrow U \ (mod\ p)$; $U \leftarrow C \cdot B$;
6. $C \leftarrow U \ (mod\ p)$; $U \leftarrow A^2$; $E \leftarrow 2D \ (mod\ p)$; $Z \leftarrow 2Z \ (mod\ p)$;
7. $B \leftarrow U \ (mod\ p)$; $U \leftarrow Y \cdot Z$; $X \leftarrow B - E \ (mod\ p)$;
   $D \leftarrow D - X \ (mod\ p)$;
8. $Z \leftarrow U \ (mod\ p)$; $U \leftarrow D \cdot A$;
9. $Y \leftarrow U \ (mod\ p)$;
10. $Y \leftarrow Y - C \ (mod\ p)$;

Point Addition Equations for secp256r1 :

1. $U \leftarrow Z^2$;
2. $A \leftarrow U \ (mod\ p)$; $U \leftarrow A \cdot Z$;
3. $B \leftarrow U \ (mod\ p)$; $U \leftarrow A \cdot P_x$;
4. $A \leftarrow U \ (mod\ p)$; $U \leftarrow B \cdot P_y$;
5. $B \leftarrow U \ (mod\ p)$; $A \leftarrow A - X \ (mod\ p)$;
6. $B \leftarrow A - Y \ (mod\ p)$; $U \leftarrow A^2$;
7. $C \leftarrow U \ (mod\ p)$; $U \leftarrow Z \cdot A$;
8. $Z \leftarrow U \ (mod\ p)$; $U \leftarrow C \cdot A$;
9. $D \leftarrow U \ (mod\ p)$; $U \leftarrow C \cdot X$;
10. $C \leftarrow U \ (mod\ p)$; $U \leftarrow B^2$;
11. $X \leftarrow U \ (mod\ p)$; $U \leftarrow D \cdot Y$; $A \leftarrow 2C \ (mod\ p)$;
    $A \leftarrow A + D \ (mod\ p)$;
12. $D \leftarrow U \ (mod\ p)$; $X \leftarrow X - A \ (mod\ p)$;
    $C \leftarrow C - X \ (mod\ p)$;
13. $U \leftarrow B \cdot C$;
14. $C \leftarrow U \ (mod\ p)$;
15. $Y \leftarrow C - D \ (mod\ p)$;

4.3.2 Hardware Architecture for ECCP in secp256k1

Here we present an overview of the hardware architecture of the scalar multiplier on 256 bit prime field with specifications $secp256k1$ [1]. The elliptic curve crypto processor (ECCP) (Figure 5) takes as input a scalar k and produces the product kP, where $P = (Px, Py)$ is a base point on the curve. For elliptic curves with efficiently computable endomorphism, the scalar multiplication consists of two steps. First the scalar is decomposed using Algorithm 3.74 of [8]. The parameters for the scalar decomposition are mentioned in Section 3.3. After decomposition, two small sized scalar k_1 and k_2 are generated, which are processed during the scalar multiplication using simultaneous point multiplication algorithm presented in [9]. We first describe the hardware unit for scalar decomposition. The decomposition block is later integrated with the Arithmetic unit.

Scalar Decomposition :

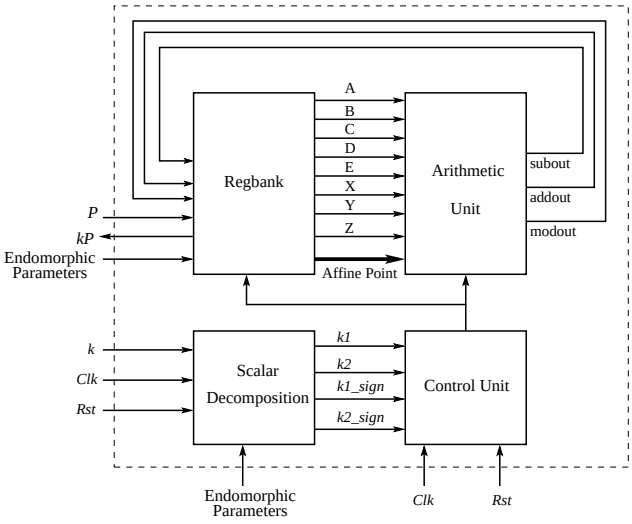

Figure 5: Block Diagram of Elliptic Curve Cryptoprocessor for $secp256k1$

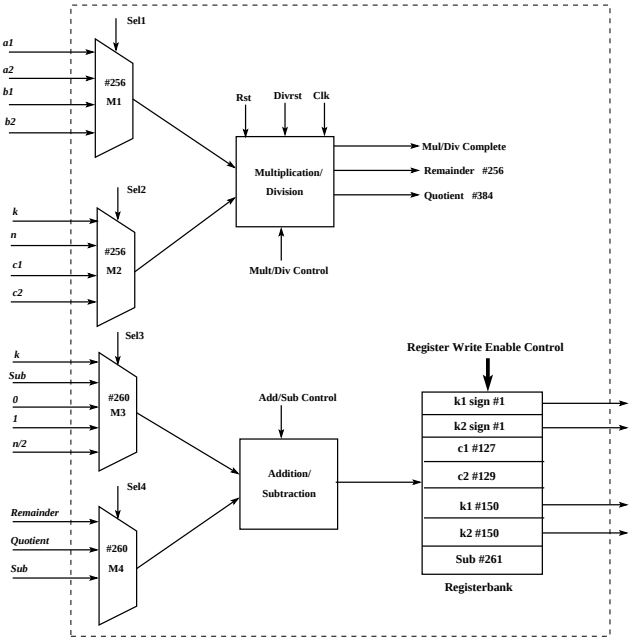

Figure 6: Scalar Decomposition Unit for $secp256k1$

The hardware architecture for the scalar decomposition unit is shown in Figure 6. Scalar decomposition involves integer multiplication, division, addition, subtraction and rounding off. Multiplication and divisions are performed using a same hardware with a distinguishing control signal *Mult/Div Control*. Similarly for integer addition and subtraction, a single adder/subtracter circuit is used with distinguishing control signal *Add/Sub Control*. The multiplication/division circuit is sequential and the number of clock cycles is equal to the number of bits in the quotient. For $secp256k1$, each multiplication or division takes 384 clock cycles. The adder/subtracter circuit is combinational. Rounding of quotient after division is performed by performing comparison between the remainder and $(n-1)/2$. When the remainder is greater than $(n-1)/2$, quotient is increased by 1. For the other case, the quotient remains unchanged. Scalar decomposition in $secp256k1$ involves six multiplications and two divisions and additional eight addition/subtraction operations. Thus total clock cycle requirement for scalar decomposition in $secp256k1$ is 3080.

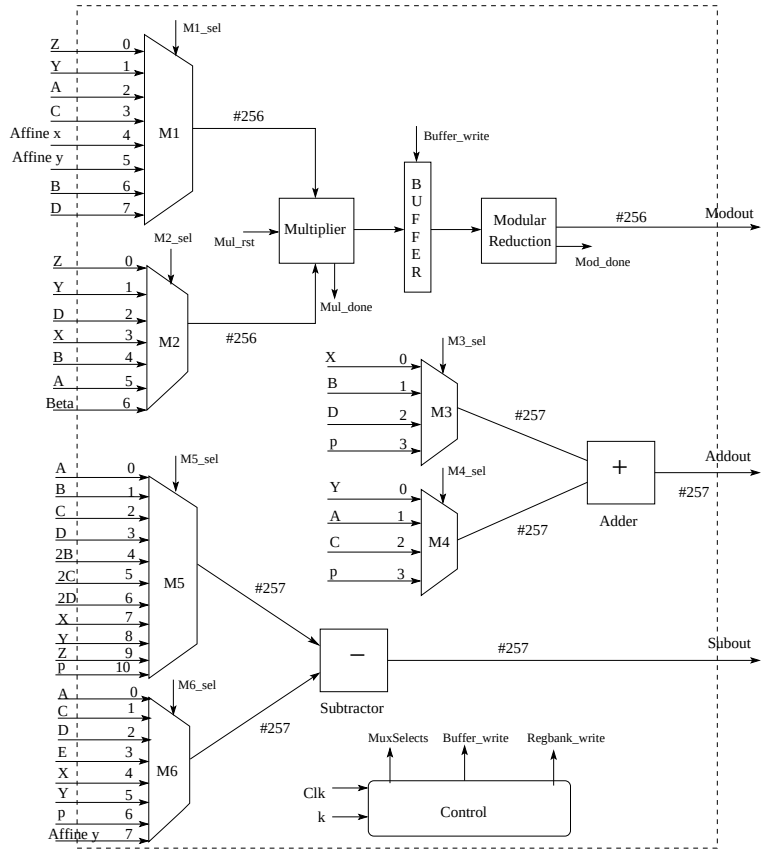

Figure 7: Arithmetic Unit for $secp256k1$

Arithmetic Unit for ECCP in $secp256k1$:

For scalar multiplication, we use left-to-right simultaneous scalar multiplication algorithm shown in Algorithm 3.77 in [8]. For simplicity we have kept the window size one. Scalar multiplication algorithm uses standard double and add formulae and starts from the MSB side of the scalar. Point doublings are performed in Jacobian projective coordinate system, while point additions are done in affine-Jacobian coordinate system [8]. We have found that for left-to-right scalar multiplications on prime fields, the above choice of coordinate systems are most efficient in reducing overall number of squaring and multiplication operations.

Since the second scalar multiplications in PSEC-KEM during encryption involves a point which is not fixed, the point is an input to the ECCP. First we compute the endomorphism $Q = \lambda P = (\beta x, y)$, which requires a single field multiplication. In the second step we compute $R = P + Q$ and the convert the result to affine coordinate system. The affine point R is stored in registerbank. During scalar multiplication, we read one key bit from $k1$ and the other key bit from $k2$. The affine point that will be added depends on the key bits of $k1$ and $k2$. The point select logic is shown in Figure 8.

Finally, at the end of point addition and doubling series, one field inversion is performed for converting projective coordinate result to affine coordinate. Arithmetic operations during scalar multiplication involves field additions, squarings, and multiplications. These finite field operations are performed in the arithmetic unit (Figure 7). Squaring operations are performed as multiplications. Thus an elliptic curve point addition requires 11 multiplications, while doubling requires 8 multiplications. Since the result of multiplication is 512 bits, modular reduction is performed to reduce it to $GF(p)$ element. To increase efficiency of the scalar multiplier, we keep the multiplier and the modular reduction units in pipeline. Since there are several field multiplications during point addition or doubling, the order of different multiplications are arranged in such a way that in most of the consecutive multiplications are independent of each other. In such cases, while modular reduction is performed over a multiplication result, the multiplier can be used to perform the next independent multiplication. Such optimizations result in lesser number of clock cycle consumption for point

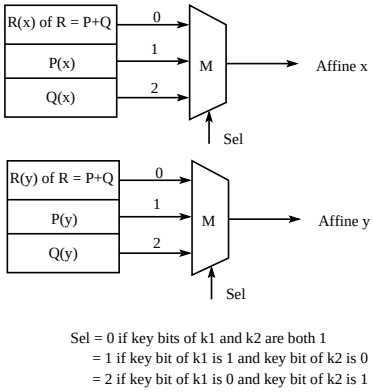

Figure 8: Affine Point Selection for Arithmetic Unit

doubling and addition. Intermediate results are stored in registers present in Regbank (Figure 5). The registers present in Regbank are A, B, C, D, E, X, Y and Z and have size either 256 bits or 257 bits. All registers are implemented using flip-flops present in FPGA slices. A control unit generates control signals in every clock cycle and drives the finite state machine for scalar multiplication. Using projective coordinates has the overhead that the result has to be converted from projective to affine coordinates. This requires an inverse to be computed followed by two multiplications. The inverse is computed BEA block. A point addition for nonzero key bit takes 119 clock cycles, while a single point double for zero key bit takes 91 clock cycles.

The overhead of using endomorphism is the requirement of scalar decomposition which consumes 3080 clock cycles for $secp256k1$. If we consider a real time application where there are several scalar multiplications, we can keep the decomposition block in pipeline. In this case, decomposition mechanism is performed in parallel with scalar multiplications and thus the decomposition cost can be ignored. For the $secp256k1$, if we assume the size of both $k1$ and $k2$ are 128 bits, then total number of clock cycles for point doubling is $91 * 128 = 11,648$ and the total number of clock cycles for point additions is $119 * 128 * 3/4 = 11,424$.

Thus the overall clock cycle requirement for point addition and doublings is $23,072$ *as compared to* $38,500$ *when endomorphism is not used.* The saving in clock cycle requirement for curves with endomorphism is thus almost 40%. More saving can be achieved is window based method is used for simultaneous scalar multiplication.

The point doubling and addition steps for the scalar multiplier are shown below. Equations that appear in the same line are performed in parallel.

Point Doubling Equations for secp256k1 :

1. $U \leftarrow Y^2$; $C \leftarrow 2X \ (mod\ p)$;
   $D \leftarrow C + X \ (mod\ p)$;
2. $B \leftarrow U \ (mod\ p)$; $U \leftarrow X \cdot D$; $B \leftarrow 2B \ (mod\ p)$;
   $C \leftarrow 2B \ (mod\ p)$;
3. $A \leftarrow U \ (mod\ p)$; $U \leftarrow C \cdot X$;
4. $D \leftarrow U \ (mod\ p)$; $U \leftarrow C \cdot B$;
5. $C \leftarrow U \ (mod\ p)$; $U \leftarrow A^2$; $E \leftarrow 2D \ (mod\ p)$; $Z \leftarrow 2Z \ (mod\ p)$;
6. $B \leftarrow U \ (mod\ p)$; $U \leftarrow Y \cdot Z$; $X \leftarrow B - E \ (mod\ p)$;
   $D \leftarrow D - X \ (mod\ p)$;
7. $Z \leftarrow U \ (mod\ p)$; $U \leftarrow D \cdot A$;
8. $Y \leftarrow U \ (mod\ p)$;

9. $Y \leftarrow Y - C \ (mod\ p)$;

Point Addition Equations for secp256k1 :

1. $U \leftarrow Z^2$;
2. $A \leftarrow U \ (mod\ p)$; $U \leftarrow A \cdot Z$;
3. $B \leftarrow U \ (mod\ p)$; $U \leftarrow A \cdot P_x$;
4. $A \leftarrow U \ (mod\ p)$; $U \leftarrow B \cdot P_y$;
5. $B \leftarrow U \ (mod\ p)$; $A \leftarrow A - X \ (mod\ p)$;
6. $B \leftarrow A - Y \ (mod\ p)$; $U \leftarrow A^2$;
7. $C \leftarrow U \ (mod\ p)$; $U \leftarrow Z \cdot A$;
8. $Z \leftarrow U \ (mod\ p)$; $U \leftarrow C \cdot A$;
9. $D \leftarrow U \ (mod\ p)$; $U \leftarrow C \cdot X$;
10. $C \leftarrow U \ (mod\ p)$; $U \leftarrow B^2$;
11. $X \leftarrow U \ (mod\ p)$; $U \leftarrow D \cdot Y$; $A \leftarrow 2C \ (mod\ p)$;
    $A \leftarrow A + D \ (mod\ p)$;
12. $D \leftarrow U \ (mod\ p)$; $X \leftarrow X - A \ (mod\ p)$;
    $C \leftarrow C - X \ (mod\ p)$;
13. $U \leftarrow B \cdot C$;
14. $C \leftarrow U \ (mod\ p)$;
15. $Y \leftarrow C - D \ (mod\ p)$;

4.3.3 Architecture for KDF

As per recommendation in document [2], mask generation function MGF1 is used as a key derivation function. MGF1 uses hash function SHA256. In the FPGA based architecture, we have used standard SHA256 available in [6]. The architecture uses only one SHA256 block. Data transfer between KDF and SHA256 happens as chunks of 32 bits. A single application of the KDF iteratively uses the SHA256 block depending on the length of output from KDF. There are two applications of KDF during encryption or decryption operations. One application of KDF requires *three* iterations and the other application requires only *one* iteration of the SHA256 block. Control signal *mode* is used as an input to the KDF block to distinguish between two different applications of KDF. An array of MUX is used to select 32 bit data input for the SHA256 block.

4.3.4 Architecture for PSEC-KEM Encryption

FPGA based architecture for encryption operation of PSEC-KEM is composed of a random number generator, KDF block, modular reduction block and ECCP. The block diagram of the architecture is presented in Figure 9. Since there are two point multiplications with two different base points (W and recommended P), two multiplexer are used at the input of ECCP for selection of the base point. The modular reduction block in the figure is of bit parallel type. Register bank is present to store outputs of KDF. In FPGA, this register bank is implemented using flip flops. During encryption operation, session key is generated from stored outputs of the KDF block. Other two outputs ECP2OSP(C1) and C2 are as per specification in the encryption operation (Section 2.1). The finite state machine (FSM) is driven by a single clock. Control block generates necessary control signals and also indicates completion after the FSM reaches the final state.

In our design, we have taken the 256 bit random string in the algorithm as an input to the design.

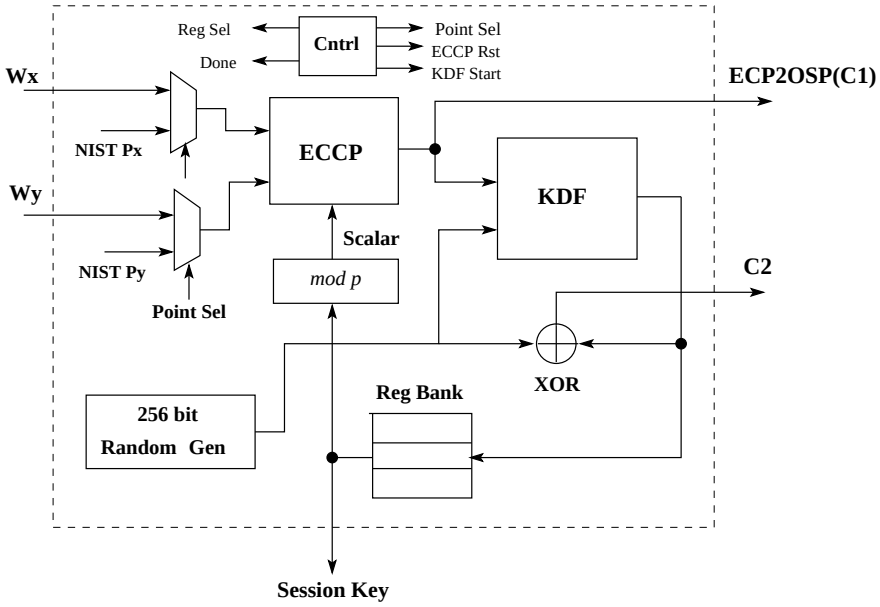

Figure 9: Architecture of PSEC-KEM Encryption for FPGA Platforms

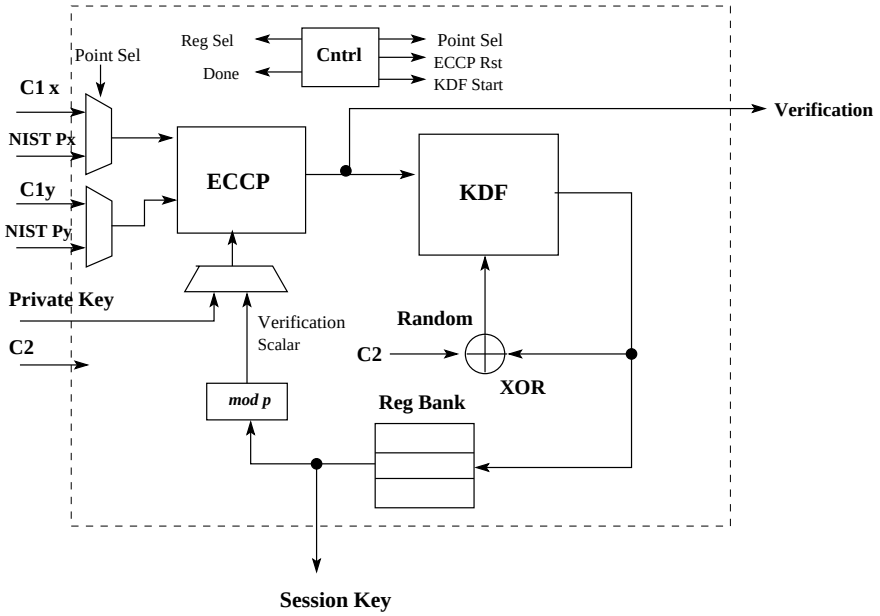

Figure 10: Architecture for PSEC-KEM Decryption for FPGA Platforms

4.3.5 Architecture for PSEC-KEM Decryption

Similar to encryption architecture, decryption architecture consists of same blocks. The architecture for decryption is presented in Figure 10. The MUX array in front of ECCP is used to select the base point. There is another MUX for scalar selection. There is a single bit output 'Verification' to indicate correctness of received string. Output of KDF is stored in register bank. The output session key is constructed by combining stored results in Register bank.

5 Results for PSEC-KEM Implementation

This section describes implementation results for software and hardware implementations of PSEC-KEM.

5.1 Software Implementation Results for PSEC-KEM

We have checked the performance of software implementation for $secp256r1$ on Intel Core 2 Duo, 2.00 GHz processor. Operating system used is Ubuntu 10.04, 32 bit. Compiler is gcc 4.4.3. Encryption and decryption operations using affine coordinate system take around 18 msec. We have checked the performance of software implementation for $secp256k1$ on the same architecture. The present implementation performs scalar multiplication using efficiently computable endomorphism. Point arithmetic are performed in affine coordinate system and scalar multiplication uses window size of 1. Encryption and decryption operations for PSEC-KEM using affine coordinate system take around 14 msec. Thus, the use of endomorphism in $secp256k1$ results in an acceleration by 29%. Further acceleration may be achieved if window size is increased.

5.2 Hardware Implementation Results

This section describes hardware implementation results for PSEC-KEM.

5.2.1 Comparisons between Scalar Multiplication in $secp256r1$ and $secp256k1$

The present hardware implementation uses only the LUTs present in a FPGA. No DSP blocks are used to perform any integer operation. Since DSP blocks are faster than LUT based field primitives, further acceleration can be achieved if they are used to implement finite field primitives. The results are taken from Xilinx ISE (Version 11.1) post-rout report. Area and timing reports are presented in Table 1. Comparison between scalar multipliers with parameters $secp256r1$ and $secp256k1$ are shown in the table. Here gain in computation time is 33%. The acceleration can be improved by using pre-computation based window methods for simultaneous scalar multiplication.

Components	Resources Utilized (Slices)	Freq (MHz) (f)	Clock Cycle (c)	Latency $msec$ (c/f)	Throughput (per sec) (f/c)
Scalar Multiplier ($secp256r1$)	9,100	44	39,500	0.9	1111
Scalar Multiplier ($secp256k1$)	9,920	43	25,700	0.6	1667

Table 1: Area and Timing for scalar multiplication in $secp256k1$ and $secp256r1$ on Xilinx Virtex V FPGA

5.2.2 Hardware Implementation Results for PSEC-KEM

Here we present area and timing report of PSEC-KEM hardwares on Xilinx Virtex V platform. Both $secp256k1$ and $secp256r1$ use almost similar design. The only difference is in the scalar multiplication and modular reduction unit. No DSP blocks have been used to design any primitive. All integer operations are performed using LUT based prime field primitives. The results are taken from Xilinx ISE (Version 11.1) post-rout report. Area and timing reports are presented in Table 2.

Components	Resources Utilized (Slices)	Freq (MHz) (f)	Clock Cycle (c)	Latency $msec$ (c/f)	Throughput (per sec) (f/c)
Encryption ($secp256k1$)	14,300	42	52,000	1.3	769
Decryption ($secp256k1$)	14,300	42	52,000	1.3	769
Encryption ($secp256r1$)	13,700	43.5	80,000	1.9	526
Decryption ($secp256r1$)	13,330	43.5	80,000	1.9	526
Encryption ($sect283r1$)	44,807	38.4	4,900	0.13	7812
Decryption ($sect283r1$)	44,623	38.2	4,900	0.13	7812

Table 2: Area and Timing for PSEC-KEM on Xilinx Virtex V FPGA

The table also contains implementation results for $sect283r1$ (binary field) and $secp256r1$ (prime field without using endomorphism). It can be seen that PSEC-KEM for elliptic curves with efficiently computable endomorphism is around 31 % faster compared to generic curves over prime fields on hardware platforms. However, $secp256k1$ is still slower compared to $sect283r1$. This happens due to delay involved due to carry propagation in prime field arithmetic. Further improvement in computation time can be achieved by incorporating DSP blocks to design prime field primitives and using pre-computation based window methods for scalar multiplication.