Implementation of PSEC-KEM in Hardware and Software

Design Document

Version 1.0

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

26th February 2011

Contents

1 Introduction **2**
1.1 Scope 2
1.2 Context 2
1.3 Constraints and Assumptions 2
1.4 Glossary 2
1.5 References 2

2 Hardware Implementation over $GF(2^{283})$ **3**
2.1 Functional Description 3
2.1.1 ES-PSEC-KEM for $GF(2^{283})$ 3
2.2 Architecture Design 4
2.2.1 Architecture for the Elliptic Curve Crypto Processor on Random Curve 4
2.2.2 Architecture for the Elliptic Curve Crypto Processor on Koblitz Curve 6
2.2.3 Architecture for KDF 6
2.2.4 Architecture for PSEC-KEM Encryption 7
2.2.5 Architecture for PSEC-KEM Decryption 8
2.3 Configuration Parameters 9

3 Software Implementation over $GF(2^{283})$ **9**
3.1 Architecture Design 9
3.2 Functional Description 9
3.3 Interface Design 9
3.4 Configuration Parameters 10

4 Software Implementation over OEF **10**
4.1 Architecture Design 10
4.2 Functional Description 12
4.3 Interface Design 12
4.3.1 Generation of Public and Private Keys 12
4.3.2 PSEC-KEM Encryption and Decryption 12
4.4 Configuration Parameters 12

1 Introduction

PSEC-KEM is a provably secure key encapsulation mechanism. It is used to realize key agreement schemes. This document describes the architecture for implementing PSEC-KEM in hardware and software.

1.1 Scope

The implementation is compatible with [2] document. The hardware and software implementation of PSEC-KEM is over the field $GF(2^{283})$. Additionally, a software implementation of PSEC-KEM in OEF is done.

1.2 Context

The implementation is used to analyze the performance of PSEC-KEM in hardware and software in binary and OEF fields.

1.3 Constraints and Assumptions

- We assume all elliptic curve points are in uncompressed format.
- Open source code is used for the hash function. The software and hardware source codes for the hash function is obtained from [8].
- The software implementations requires Linux operating system and has been tested with gcc-4.4.3. Also, unless otherwise stated we assume that the OS and the processor is 64 bit.

1.4 Glossary

OEF	Optimal Extension Field
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

1.5 References

[1] NTT Information Sharing Platform Laboratories, NTT Corporation. *Standars 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. *Standars 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] D.V.Bailey. *Computation in Optimal Extension Fields.* MS Thesis, May 2000.

[7] M.K.Lee, K.T.Kim, H.Kim, and D.K.Kim. *Efficient Hardware Implementation of Elliptic Curve Cryptography over* $GF(p^m)$. WISA 2005, LNCS 3786, pp. 207-217, 2006.

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

[9] C. Rebeiro and D. Mukhopadhyay, "Power Attack Resistant Efficient FPGA Architecture for Karatsuba Multiplier," in VLSID 08: Proceedings of the 21st International Conference on VLSI Design, IEEE Computer Society, 2008, pp. 706–711.

[10] C. Rebeiro, S.S. Roy, D.S. Reddy and D. Mukhopadhyay, "Revisiting the Itoh-Tsujii Inversion Algorithm for FPGA Platforms", IEEE Transactions on VLSI Systems, vol. PP Issue:99 (pre-print).

[11] Jerome A. Solinas, "Effecient Arithmetic on Koblitz Curves", Design, Codes and Cryptography, 2009, pages 195-249.

[12] Billy Bob Brumley and Kimmo U. Järvinen, "Conversion Algorithms and Implementations for Koblitz Curve Cryptography", IEEE Transactions on Computers, 2010, pages 81-92.

2 Hardware Implementation over $GF(2^{283})$

2.1 Functional Description

2.1.1 ES-PSEC-KEM for $GF(2^{283})$

The encryption and decryption algorithms for ES-PSEC-KEM are specified in [2]. Here we present the algorithms for the field $GF(2^{283})$. The irreducible polynomial defined over $GF(2^{283})$ is $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$. As per document [2], we have taken the octet length parameters $pLen = 28$, $hLen = 32$ and $keyLen = 32$.

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 $\alpha := OS2FEP(t)\ mod\ f(x)$.
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.

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 $\alpha := OS2IP(t)\ mod\ f(x)$.
9. Check $C_1 := \alpha P$. If it holds output k. Otherwise, assert INVALID and stop.

2.2 Architecture Design

In this section we present FPGA based architecture for encryption and decryption operations in PSEC-KEM. The main components of the hardware are, an elliptic curve crypto processor (ECCP) and key derivation function (KDF). The ECCP does scalar multiplication on the NIST pseudo random curve defined over $GF(2^{283})$ [1]. We first describe the architectures of the ECCP and the KDF and then the architecture of the PSEC-KEM encryption and decryption blocks.

2.2.1 Architecture for the Elliptic Curve Crypto Processor on Random Curve

Here we present an overview of the hardware architecture of the random curve [1] based scalar multiplier defined over $GF(2^{283})$. The elliptic curve crypto processor (ECCP) (Figure 1) takes as input a scalar k and produces the product kP, where $P = (Px, Py)$ is a point of the curve. Since the second scalar multiplications during encryption involves a point which is not fixed, the point is an input to the ECCP. The curve constant is stored in the ROM and loaded into registers during initialization. The ECCP implements the scalar multiplication algorithm using the elliptic curve double and add formulae. Point doubling and point additions are done in Lopez-Dahab co-ordinate system. Arithmetic operations during scalar multiplication involves field additions, squarings, and multiplications (Figure 2). The field multiplier in our design follows hybrid Karatsuba algorithm [9]. The hardware for multiplication is the biggest, therefore the ECCP can afford to have only one finite field multiplier. Several adders and squarers can be used as they contribute marginally to the latency and area of the processor. The elliptic curve point addition has 8 multiplications, therefore with one multiplier (which is capable of doing one multiplication per clock cycle) it would require 8

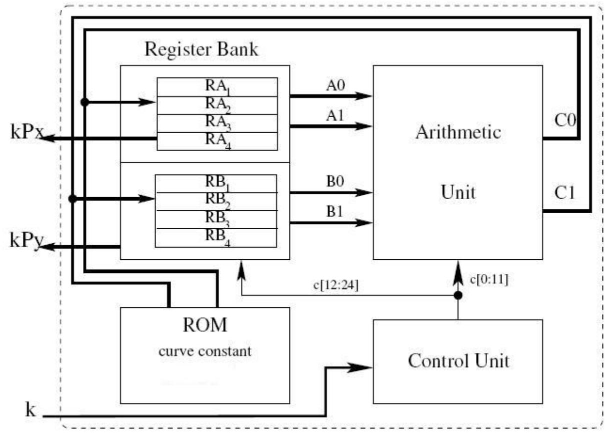

Figure 1: High Speed Compact Elliptic Curve Cryptoprocessor for FPGA Platforms

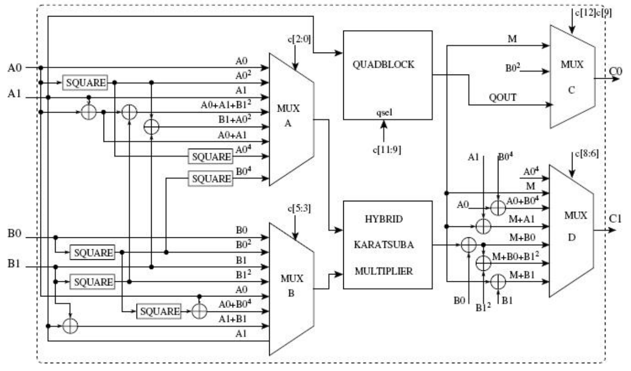

Figure 2: Arithmetic Unit

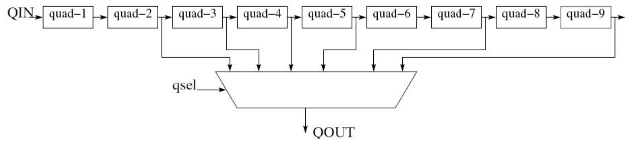

Figure 3: Cascaded Quad Blocks with Multiplexer

clock cycles. Similarly point doubling, which requires 4 multiplications, would require a minimum of 4 clock cycles. The design of the arithmetic unit is optimized so that the addition and doubling takes the minimum required clock cycles. The arithmetic unit in the ECCP has two outputs. At every clock cycle, at least one of the outputs is derived from the multiplier. This ensures that the multiplier is used in every clock cycle. In order to generate two outputs the arithmetic unit requires at least four input lines. The data on the four input lines is read from registers. The registers are implemented using the FPGAs dual ported distributed RAM. Each dual ported RAM has two address lines, two output data lines, and one input data line, therefore to feed the four input arithmetic unit at least two dual ported RAMs (RA and RB) are required. Each dual ported RAM, called a register file, implements four 283 bit registers and the two register files are collectively known as a register bank. The control unit generates a 25 bit control word every clock cycle. This control word selects the four registers whose data would be read, selects the inputs to the multiplier using multiplexers MUXA and MUXB and selects the output of the arithmetic unit through MUXC and MUXD. The control word also determines the register where the result gets written into. Using LD 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 using Itoh-Tsujii inversion algorithm. In this architecture, Itoh-Tsujii inversion algorithm uses quad operation ($\alpha \rightarrow \alpha^4$) instead of squaring ($\alpha \rightarrow \alpha^2$)

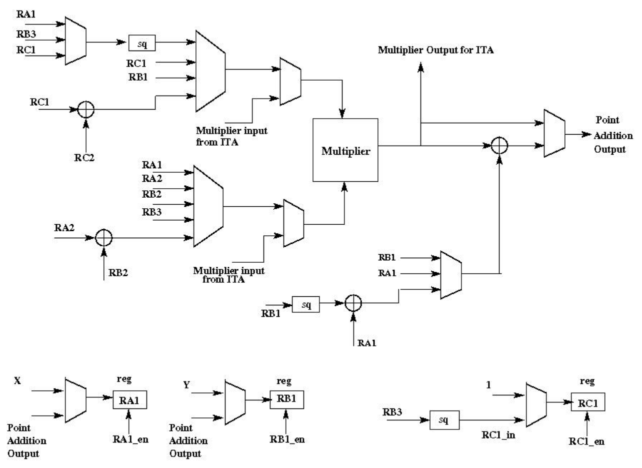

Figure 4: Point Addition Unit

[10]. The inversion circuit uses the multiplier of the ECCP. Repeated quading is performed using a cascade of quad circuites (Figure 3).

2.2.2 Architecture for the Elliptic Curve Crypto Processor on Koblitz Curve

In this section we present an architecture for a Koblitz curve based ECCP in $GF(2^{283})$. The architecture does scalar multiplication using $\tau-$adic NAF method [11]. One problem in $\tau-$adic NAF method is that, the $\tau-$adic NAF representation of the scalar has length close to double the length of it's binary representation. In [11], reduction algorithm was presented which results in a $\tau-$adic NAF which has length close to the binary length of the scalar. In [12], efficient FPGA based reduction architecture was presented which uses Lazy Reduction algorithm. The same hardware is used to generate $\tau-$adic NAF bits sequentially. In Figure 5, we present our new architecture for lazy reduction of the scalar and $\tau-$adic NAF generation. Our reduction architecture uses same amount of hardware as in [12], but generates *two* $\tau-$adic NAF bits per clock cycle. In $\tau-$adic NAF method, point squaring is performed in every iteration. Since, two $\tau-$adic NAF bits are generated per clock cycle, we use a cascade of two squarers (Figure 6). In $\tau-$adic NAF, density of nonzero bits is nearly $\frac{1}{3}$, and thus point addition is required only for $\frac{1}{3}$ bits of the entire $\tau-$adic NAF. In the scalar multiplication architecture, there is an efficient control unit which controls point addition and point squaring. Point addition is done in Lopez-Dahab coordinate system and requires eight clock cycles. During point addition, the control unit continues point squaring for next 'zero' bits of the $\tau-$adic NAF. When a nonzero $\tau-$adic NAF pair is encountered during point addition, point squaring and $\tau-$adic NAF generation are suspended. When point addition starts for the new value, $\tau-$adic NAF generation and point squaring are resumed. Figure 4 describes the architecture for point addition block. The multiplier inside this point addition unit is shared with inversion unit (not shown in the figure). The inversion unit does field inversion of the final Z coordinate using quad Itoh-Tsujii algorithm [10].

2.2.3 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 [8]. The hardware architecture of KDF is present in Figure 7. The architecture uses only one SHA256 block. An array of multiplexer is used for different inputs to the SHA256 block. 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. 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. Output from the KDF block is of size 256 bits.

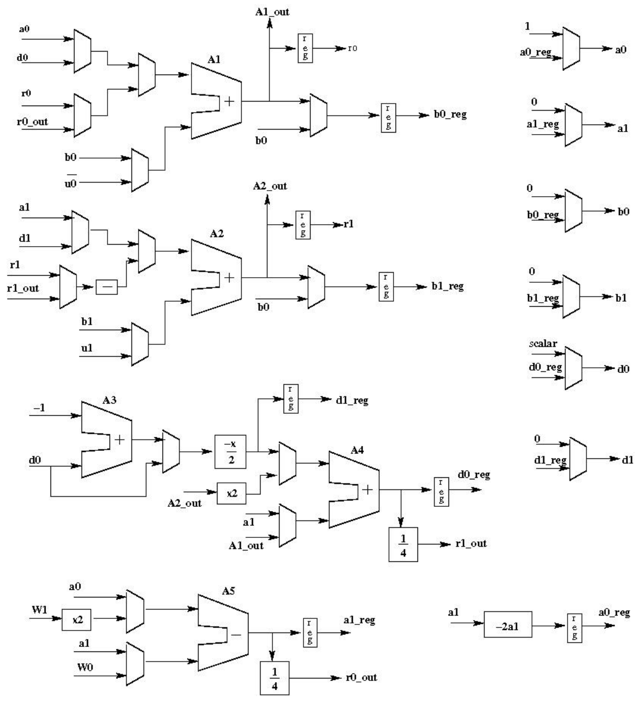

Figure 5: Scalar Reduction and $\tau-$adic NAF generation Unit

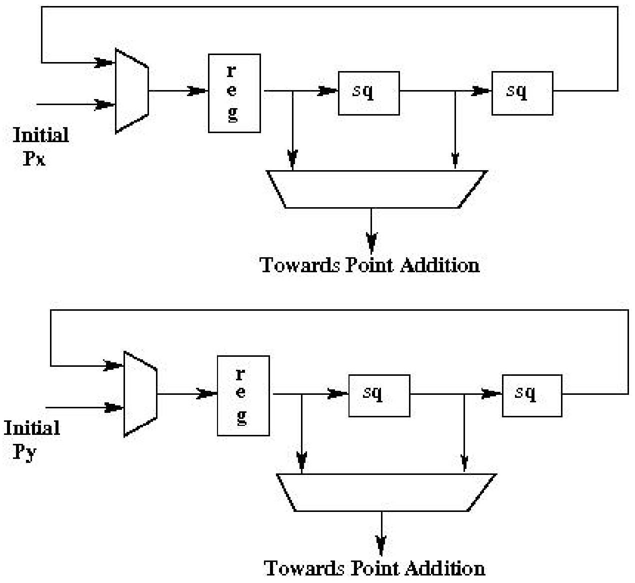

Figure 6: Point Squaring Unit

2.2.4 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 8. Since there are two point multiplications with two different base points (W and [2] 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 Algorithm (2.1.1).

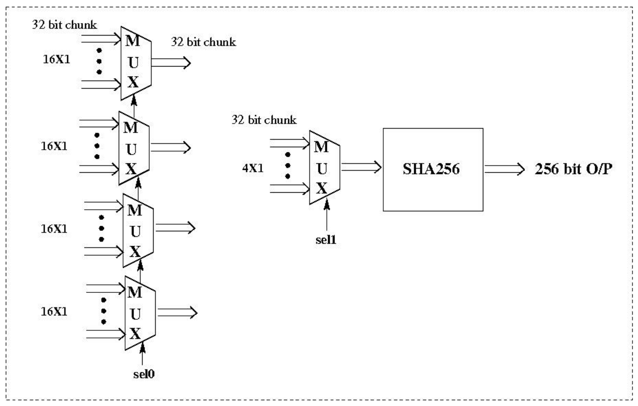

Figure 7: Architecture of KDF for FPGA Platforms

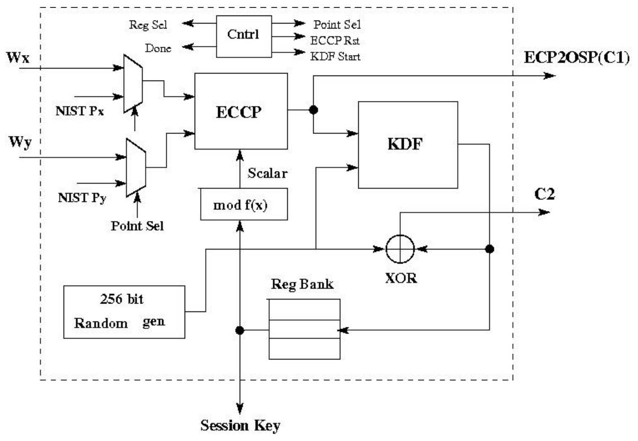

Figure 8: Architecture of PSEC-KEM Encryption for FPGA Platforms

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.

2.2.5 Architecture for PSEC-KEM Decryption

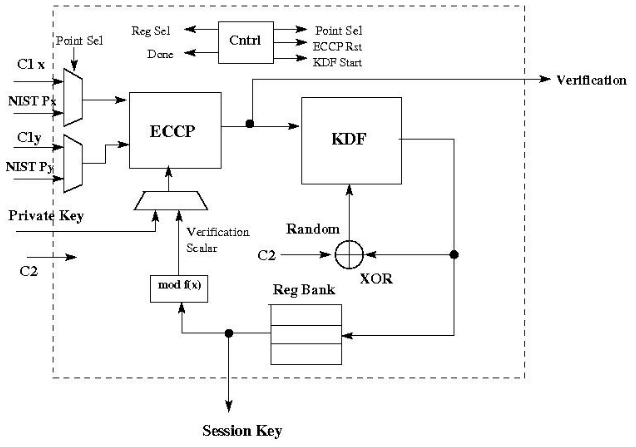

Figure 9: Architecture for PSEC-KEM Decryption for FPGA Platforms

Similar to encryption architecture, decryption architecture consists of same blocks. The architecture for decryption is presented in Figure (9). 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.

2.3 Configuration Parameters

In the design, configuration parameters are: base point and elliptic curve constants. If configuration parameters change for the same finite field, then the parameters must be redefined and the design must be re-implemented.

3 Software Implementation over $GF(2^{283})$

In this section we present the implementation aspects of PSEC-KEM which uses random and Koblitz curves over the field $GF(2^{283})$.

3.1 Architecture Design

- **Random Number Generator :** This is used during the encryption process. The implementation uses the standard unix *random()* function for the purpose.
- **Hash Function :** SHA-256 is used as the hash function. Open source software obtained from [8] was used in the design.
- **Scalar Multiplication :** In this section we present software design for encryption and decryption operations in PSEC-KEM. Our main focus was on efficient implementation of finite field primitives. The main components of the software are a elliptic curve scalar multiplier and hash function. The hash function is standard SHA256 and the code for SHA256 was obtained from [8]. Elliptic curve scalar multiplier is the most important part of the software. Efficiency of the scalar multiplier depends on the efficiency of finite field primitives and on the form of scalar. For pseudo random curves, scalar multiplication is done using double and add method. Both point doubling and point additions are performed in Lopez-Dahab co-ordinate system. Elliptic curve point addition and doubling involve finite field multiplication. Field multiplication is a costly operation and thus software implementation requires very efficient field multiplier. Our field multiplier follows window technique with window size 4 bits. Use of Lopez-Dahab co-ordinate system in scalar multiplication requires one field inversion at the end of scalar multiplication. We have used 'Extended Euclidean Algorithm' for finite field inversion.

  We have also implemented the protocol using Koblitz curve defined over $GF(2^{283})$. The scalar is converted into $\tau-$adic NAF before scalar multiplication. We have optimized the finite field primitives specific to 64 bit processor. The implementation does not involve any parallelism.

3.2 Functional Description

Please refer Section 2.1.

3.3 Interface Design

The software for PSEC-KEM is a menu driven program. User has four options:
(i) Private key generation
(ii) Public key generation
(iii) Encryption operation for PSEC-KEM
(iv) Decryption operation for PSEC-KEM.

The operations does the following:

1. Private key is generated as a random scalar of size 283 bits.
2. Public key generation does scalar multiplication sP to produce public key W.
3. Encryption operation does session key generation and session key encapsulation.
4. Decryption operation asks for the private key and produces the session key from it's encapsulated form. It also does verification operation.

3.4 Configuration Parameters

Parameter	Where Stored	Comment
_polylen	FINITE.H	stores the length of irreducible pentanomial.
_dpolylen	FINITE.H	stores the length of field multiplier output.
hLen	FINITE.H	PSEC-KEM parameter
pLen	FINITE.H	PSEC-KEM parameter
keyLen	FINITE.H	PSEC-KEM parameter
CURVECONST_A	FINITE.H	The curve constant A for the random elliptic curve [1]
CURVECONST_B	FINITE.H	The curve constant B for the random elliptic curve [1]
CURVE_BASEPT_X	FINITE.H	The X coordinate base point for the curve [1]
CURVE_BASEPT_Y	FINITE.H	The Y coordinate base point for the curve [1]

4 Software Implementation over OEF

In this section we present the implementation aspects of PSEC-KEM over OEF.

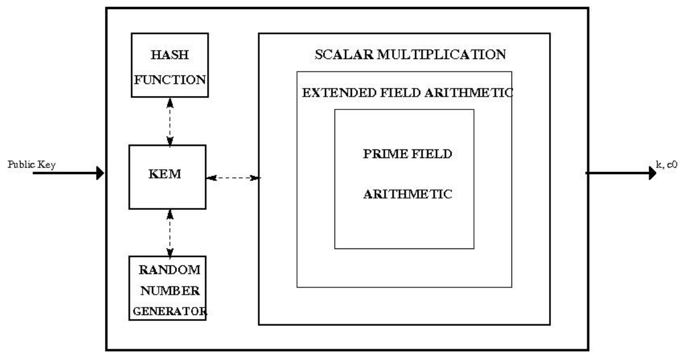

Figure 10: Architecture for Sofware Implementation of PSEC-KEM over OEF

4.1 Architecture Design

An OEF is a finite field $GF(p^m)$ that satisfies the following properties

1. $p = 2^n - c$, where $log_2|c| \leq n/2$
2. An irreducible binomial $f(x) = x^m - w$, where $(w \in GF(p))$

The software block diagram of PSEC-KEM over OEF is as shown in Figure 10. The various modules in the architecture is discussed below:

- **KEM :** The KEM module implements the algorithm for encryption according to the Algorithm ES-PSEC-KEM-ENCYPT and ES-PSEC-KEM-DECRYPT defined in Section 5.2 of [2]. The implementation of the functions $OS2IP$, $I2OSP$, $ESP2OSP$, $PESP2OSP$, and $OS2ECPP$ that are required for encryption and decryption are as per the Algorithms specified in [3] and [5]. These algorithms require multi-precision arithmetic, hence they are implemented using the *gmp* library.
- **Random Number Generator :** The random number generation is required to seed the hash function. Currently the POSIX $random()$ libary call is used for the purpose.
- **Hash Funtion :** Refer to Software Implementation over $GF(2^{283})$ Section 3.1.
- **Scalar Multiplication :** The elliptic curve has the form

$$Y^2 = X^3 + AX + B \tag{1}$$

where $A, B \in GF(p^m)$ and $4A^3 + 27B^2 \neq 0$. The values of the curve constants (A and B) and the basepoint are taken from [3] and [7]. The fields that have been tested are listed in Table 4.4. In the table, the field is represented in the form $GF(p^m)$ with irreducible polynomial $f(x) = x^m - c$.

Name	p	m	w	Reference
ECP31M07K	$2^{31} - 1$	7	3	[7]
SECO305R	$2^{61} - 1$	5	3	[3]
SECO427R	$2^{61} - 1$	7	3	[3]

Table 1: OEF's Supported by the PSEC-KEM Implementation

Standard double-and-add algorithm is used to implement the scalar multiplication. It uses Jacobian projective coordinates for representation of the points on the curve.

- **Extended Field Operations :** The extended field has the form $GF(p^m)$ with irreducible polynomial $f(x) = x^m - c$. The finite field arithmetic is implemented as follows:
  - *Addition* and *subtraction* are polynomial arithmetic with coefficients in the field $GF(p)$.
  - *Multiplication* is done using the classical algorithm for polynomial multiplication. The *modular reduction* required at the end of the multiplication uses the technique presented in Section 4.2.1 in [6].
  - *Inversion* uses the Itoh-Tsujii technique as described in Chapter 5 in [6].
- **Prime Field Operations :** The arithmetic in the extended field internally uses the prime field arithmetic routines. The implementation of these are described below.
  - *Addition* and *subtraction* are done modulo the prime p.
  - *Multiplication:* Here we assume a 32 bit OS and processor. Depending on the size of the field, there are two techniques which are adopted. Let the multiplicands be a, $b \in GF(p)$ and $c = ab \bmod p$.
    - When $p < 2^{32}$: We compute $c' = a \cdot b$ and $c'' = \lfloor c'/p \rfloor$. Note that d is a 64 bit value. The required product is $c = c' - c''$. Thus computing the product requires one multiplication and one division.
    - When $2^{32} < p < 2^{64}$: We consider the multiplicands a and b as comprising of smaller values. ie. $a = (a_h|a_l)$ and $b = (b_h|b_l)$, where a_l and b_l are 32 bit values. The product before the modular reduction is given by $c' = a_h b_h * 2^n + (a_h b_l + b_h a_l)2^{32} + a_l b_l$. The modular reduction to compute $c = c' \bmod p$ is obtained by the technique in Section 4.2.3 in [6].
  - *Inversion :* Inversion in prime field is implemented using the extended Euclidean algorithm.

4.2 Functional Description

Refer to Section 2.1.

4.3 Interface Design

4.3.1 Generation of Public and Private Keys

The key generation is done as follows:

1. A random scalar s is generated of size less than p^m. This is the private key.
2. For the specified base point P, the scalar product sP is computed. This is the public key.

To generate the public and private key execute,

```
$./genkey
```

This would create a file for private key (KEY.PRIV) and public key (KEY.PUB).

4.3.2 PSEC-KEM Encryption and Decryption

The input to the encryption is the file KEY.PUB. The output is the encapsulated key, is stored in the file ENCKEY. For the decryption, the input is the file ENCKEY.

For encryption run the executable

```
$ ./kem enc
```

For decryption run the executable

```
$ ./kem dec
```

4.4 Configuration Parameters

Parameter	Where Stored	Comment
FIELD_M	*fftypes.h*	stores the degree of the irreducible binomial.
FIELD_W	*fftypes.h*	stores the irreducible binomial coefficient.
FIELD_P	*fftypes.h*	the prime number
MULALGO0	*fftypes.h*	Used for finite field multiplication $p < 2^{32}$
MULALGO1	*fftypes.h*	Used for finite field multiplication $p > 2^{32}$

Each of the supported fields have a corresponding header file (*seco305r.h*, *seco427r.h*, and *ecp31m07k.h*) in which the curve parameters are defined. Table 4.4 gives the macros that define the curve parameters.

Parameter	
CURVECONST_A	The curve constant A
CURVECONST_B	The curve constant B
CURVE_BASEPT_X	The X coordinate basepoint
CURVE_BASEPT_Y	The Y coordinate basepoint