Implementation of PSEC-KEM using C for Prime Fields

Software Manual

Indian Institute of Technology Kharagpur
West Bengal

January 2012

Contents

1 Introduction **2**

2 Requirements **2**

3 Operation **2**

3.1 Compilation and Execution 2

3.2 Performing Key Encapsulation 2

4 Internals **3**

4.1 The `src` folder 3

4.2 Functions and Dependencies 3

4.2.1 `fp.c` and `include/fp.h` 3

4.2.2 `ec.c` and `include/ec.h` 4

4.2.3 `psec-kem.c` and `include/psec-kem.h` 5

4.2.4 `main2.c` 6

5 Screenshots **8**

©NTT Corporation, IIT Kharagpur, 2012

1 Introduction

This is a documentation and user-manual for the C implementation of the PSEC Key Encapsulation Mechanism.

2 Requirements

Following programs need to be installed on the 32-bit UNIX machine

- A C compiler
- GNU Multi Precision Library (GMP)
- GNU Make

3 Operation

3.1 Compilation and Execution

1. Untar the tar archive using

   ```
   $ tar -xvf psec-kem.tar
   ```

2. Browse to the `src/include` folder and open the file `psec-kem.h`

3. The various options are as follows. Select the appropriate ones.

   ```
   #define VERBOSE 0
   #define TIMED 1
   #define ITER 30 // no. of iterations for time computation
   #define USE_PROJECTIVE_JACOBIAN 1
   ```

4. Browse to the `src` folder in the new `psec-kem` folder that is formed

5. Compile `psec-kem` using

   ```
   $ make psec
   ```

6. Run the executable using

   ```
   $ ./psec
   ```

3.2 Performing Key Encapsulation

On running the executable you should see a prompt like this

```
yatendra@yatendra-lptp:/media/yat/acad/designlab/psec-kem/src$ ./psec
using projective Jacobian coordinate system...
PSEC-KEM options are :
1. 256 bit Private Key Generation
2. Public key W(x,y) generation
3. Encryption operation of PSEC-KEM
4. Decryption operation of PSEC-KEM

Enter choice :
```

Enter the appropriate choice and follow the appropriate instructions.

4 Internals

This section describes the details of the code organization.

4.1 The `src` folder

The `src` folder contains the following files

`fp.c` : Contains the functions for finite field operations

`ec.c` : Contains the elliptic curve operations : point-addition and scalar-multiplication using jacobian and affine coordinate systems

`psec-kem.c` : Contains the psec-kem modules like KDF, Encryption and Decryption

`main1.c` : Contains a test module to test the scalar-multiplication

`main2.c` : Contains the module to perform psec-kem

`mrshs256.c, miracl.h, mirdef.h` : The hash function used in KDF are defined here

`Makefile` : The GNU Makefile

`include` **directory:** This directory contains

- `fp.h` : Contains the prototypes of the functions used in the file `fp.c`
- `ec.h` : Contains the prototypes of the functions used in the file `ec.c`
- `psec-kem.h` : Contains the prototypes of the functions used in the file `psec-kem.c`

4.2 Functions and Dependencies

This subsection defines the various functions in the files described in 4.1.

4.2.1 `fp.c` and `include/fp.h`

Description: This file contains the field operations
Dependency: Only dependencies are the GMP library (`gmp.h`) and `stdio.h`
Variables:

`fp_elt` : This is a large integer `mpz_t` representing an element of $\mathbb{F}_p$

Functions:

`fp_init(fp_elt e)` : Initialises the large integer `mpz_t` for use. To be done after every declaration of `e`

`fp_set(fp_elt e_dst, fp_elt e_src)` : Copies the element `e_src` to `e_dest`

`fp_clear(fp_elt e)` : Clears the memory used by the large integer `e`

`fp_print(fp_elt e)` : Prints the large integer `e`

`int fp_cmp(fp_elt e1, fp_elt e2)` : Compares two large integers `e1` and `e2`. Returns `0` if equal, otherwise nonzero value is returned

`int fp_cmp_ui(fp_elt e, unsigned int t)` : Compares the large integer `e` to the unsigned integer `t` and returns `0` if equal, non-zero otherwise

`fp_add(fp_elt e, fp_elt e1, fp_elt e2, mpz_t P)` : Adds the two large integers `e1` and `e2` modulo the large prime `P` and returns output in `e`

`fp_sub(fp_elt e, fp_elt e1, fp_elt e2, mpz_t P)` : Subtracts `e2` from `e1` modulo the prime `P` and returns output in `e`

`fp_mul(fp_elt e, fp_elt e1, fp_elt e2, mpz_t P)` : Multiplies `e1` to `e2` modulo prime `P` and returns output in `e`

`fp_mul_ui(fp_elt e, fp_elt e1, unsigned int t, mpz_t P)` : Mulitplies element `e1` to unsigned integer `t` modulo the prime `P` and returns the result in `e`

`fp_exp(fp_elt e, fp_elt e1 , unsigned int m, mpz_t P)` : Returns modular exponentiation $e1^m$ modulo `P` and returns the output in `e`

`fp_reduce(fp_elt e1, mpz_t P)` : Reduces `e1` modulo `P`. Used by other functions to reduce final result of addition, multiplication and subtraction.

`fp_inverse(fp_elt e, fp_elt e1, mpz_t P)` : Due to this `e` stores the modular inverse of `e1` modulo `P`

4.2.2 `ec.c` and `include/ec.h`

Description: This file contains the elliptic curve operations for both affine and jacobian projective coordinate systems
Dependency: `fp.h`, `stdlib.h`, `assert.h`
Variables:

`pointTyp` : An affine point containing two large `fp_elt`s as x and y coordinates

`jacobianPointTyp` : A Jacobian-coordinate point containing three `fp_elt`s x, y and z

`ec_point` : Pointer to `pointTyp`

`ec_jacobian_point` : Pointer to `jacobianPointTyp`

Functions:
The functions are organised as follows, based on the prefix (table 1) and suffix(table 2). Other functions

Table 1: Functions on basis of prefix

Prefix	Purpose
`ec_`	Basic initialisation, copy, comparison, clear and print functions for points in *affine coordinates*
`ec_affine_`	Addition, Scalar-Multiplication operations for points in *affine coordinates*
`ec_jacobian_`	Outer wrapper for jacobian coordinate operations. Input and output are in *affine coordinates* but the operation is done using jacobian projective coordinates. These functions call functions prefixed with `__ec_jacobian_`
`__ec_jacobian_`	Input and output are points are in *jacobian projective coordinate system.* Also includes the basic initialisation, copy, comparison, clear and print functions

used for the affine and jacobian transformations are

`affine2jacobian(ec_jacobian_point dest, ec_point src)` : Converts the affine point `src` to jacobian point `dest`

`acobian2affine(ec_point dest, ec_jacobian_point src, mpz_t P)` : Converts the jacobian point `src` to affine point `dest`

Table 2: Functions on basis of suffix

Suffix	Purpose
`init(pt)`	Initialises the point `pt`. To be done after declaring the point.
`copy(dst,src)`	Copies the elliptic curve point coordinates of `src` to `dst`
`isequal(p1,p2)`	Returns `0` if the points `p1` and `p2` are not the same
`print(p,char* lbl)`	Prints the point `p` and puts a label `lbl` to identify the point
`clear(p)`	Clears the space occupied by the point `p`
`add(s,a,b,P,A,B)`	Performs addition of points `a` and `b` and saves result to `s`. Other inputs are `P` which is the order of the prime field over which the curve is defined and `A`, `B` which are the curve parameters.
`adddistinct(s,a,b,P,A,B)`	Called by `add` to add points if `a` $\neq$ `b`
`double(s,a,P,A,B)`	Called by `add` to add points if `a` $=$ `b`
`mul(r,p,k,P,A,B)`	Computes the scalar multiplication `kp` using the Montgomery ladder for affine points and double-and-add for jacobian points, where `k` is the scalar and stores the result in `r`

4.2.3 `psec-kem.c` and `include/psec-kem.h`

Description: This file contains the implementation of the psec-kem and depends on the field library `fp.h` and elliptic curve library `ec.h`
Dependency: `fp.h`, `ec.h`, `time.h`, `string.h`
Variables:
The global variables are described below.

`mpz_t A,B,p :` The curve is defined as $y^2 = x^3 + \mathtt{A}x + \mathtt{B}$ over the field $\mathbb{F}_p$.

`ec_point P :` This is the base point used to generate the public key from the secret key.

`mpz_t s :` The secret key which is a large integer.

`ec_point W :` The public-domain key which is a point on the elliptic curve

The compiler directives are described below.

`VERBOSE :` Set this to `1` to get detailed output during encryption and decryption. The notations for points are the same ones used in the design document.

`TIMED :` Set to `1` to output running times for encryption and decryption over `ITER` runs.

`ITER :` Set to the number of iterations over which the times are to be recorded. Irrelevant if TIMED is set to 0.

`USE_PROJECTIVE_JACOBIAN :` Set to `1` to use projective jacobian coordinates which is faster.

NOTE: Octet string is defined to be an array of characters (not a String because they may not be ascii characters and are not NULL-terminated). For example, the array $\{90, 34, 0, 132, 45\}$ is an octet string of length 5. Hence octet string is not same as a string. Hexadecimal string however is a string which is used to represent an octet string using hexadecimal characters. Therefore an octet string of length 5 can be represented using 10 hexadecimal characters. Because an octet represents 8 bits, whereas hexadecimal character represents 4 bits.

Functions: The functions are divided into Simple integer-character conversions(table 3), Octet String, Integer and EC Point conversions(table 4), Octet string operations(table 5) and PSEC-KEM primitives(table 6).

Table 3: Simple integer-character conversions

Function	Purpose
`getdigit(char c)`	Returns the integer in 0-15 corresponding to the input hexadecimal character in 0-F
`gethexchar(int digit)`	Returns the hexadecimal character 0-F corresponding to input integer in 0-15
`getoctet(char* hex)`	Input is two hexadecimal characters (i.e. 8 bits). Ouput is character corresponding to those 8 bits. **Example:** `getoctet("5a") = (char)90`

Table 4: Octet String, Integer and EC Point conversions

Function	Purpose
`int2os(char* os, mpz_t n, long int outLen)`	Converts large integer `n` to octet string `os` of octet length `outLen` padding with zeros when necessary
`hexstr2os(char* os, char* hex , long int outLen)`	Converts the hexadecimal string `hex` to octet string `os` of octet length `outLen`
`os2int(mpz_t n, char* os, int inLen)`	Converts octet string `os` of length `inLen` to large integer removing the 0-padding if any
`ec2os(char* os, ec_point p)`	Converts EC Point `p` to octet string `os` using *Uncompressed* form
`os2ec(ec_point p, char* os)`	Converts octet string `os` in *Uncompressed* form to EC Point `p`

Table 5: Octet string operations

Function	Purpose
`os_hash(char* hash, char* in, int lenin)`	Inputs an octet string `in` of length `lenin` and outputs the `hash` as octet string. Requires the hash functions from `mrshs256.c, miracl.h` and `mirdef.h`
`os_concat(char* dest, char* src1, char* src2, int len1 , int len2)`	Concatenates octet strings of given lengths and outputs the concatenated string in `dest`. The space for `dest` needs to be allocated by the caller.
`os_xor(char* ros, char* in1, char* in2, int len)`	Performs XOR operation between two octet strings of equal length `len` and stores the output in `ros`

Table 6: PSEC-KEM primitives

Function	Purpose
`kdf(char* mask, char* M, int lenM, long int n)`	Outputs a `mask`(octet string) of octet length `n` from the input seed `M` of length `lenM` octets
`psec_kem_encrypt(char* k , char* c0)`	The PSEC-KEM Encryption function that outputs `k`(The session key) and `c0`(Public Message) It uses the global variables `A, B, p, P, W`
`psec_kem_decrypt(char* k , char* c0)`	The PSEC-KEM Decryption function that inputs a message `c0`. If it returns 0, then the output session key `k` is set to the string `"INVALID"`. If the return value is 1, the output session key `k` contains the session key, which can be printed in the calling function.

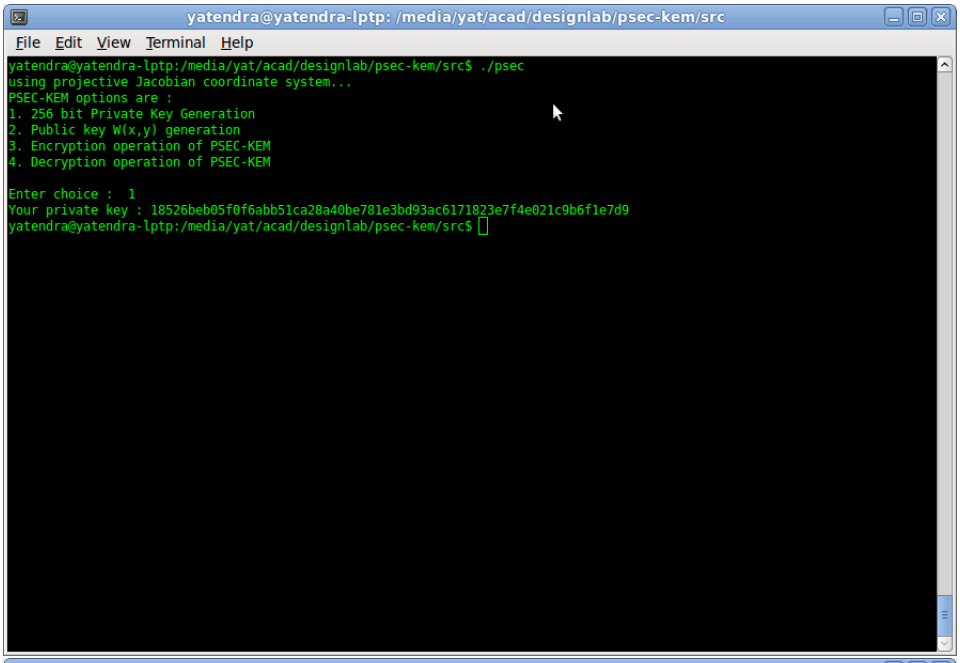
4.2.4 `main2.c`

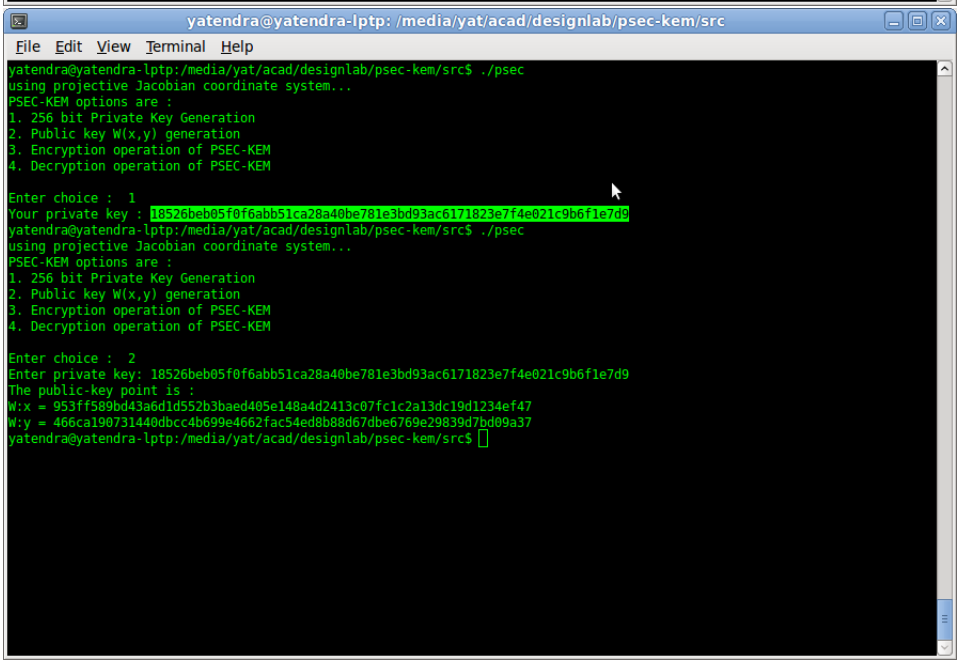
Description: This file is used to test the PSEC-KEM libraries
Dependency: `psec-kem.h`
Working: The following tasks are performed in this program.

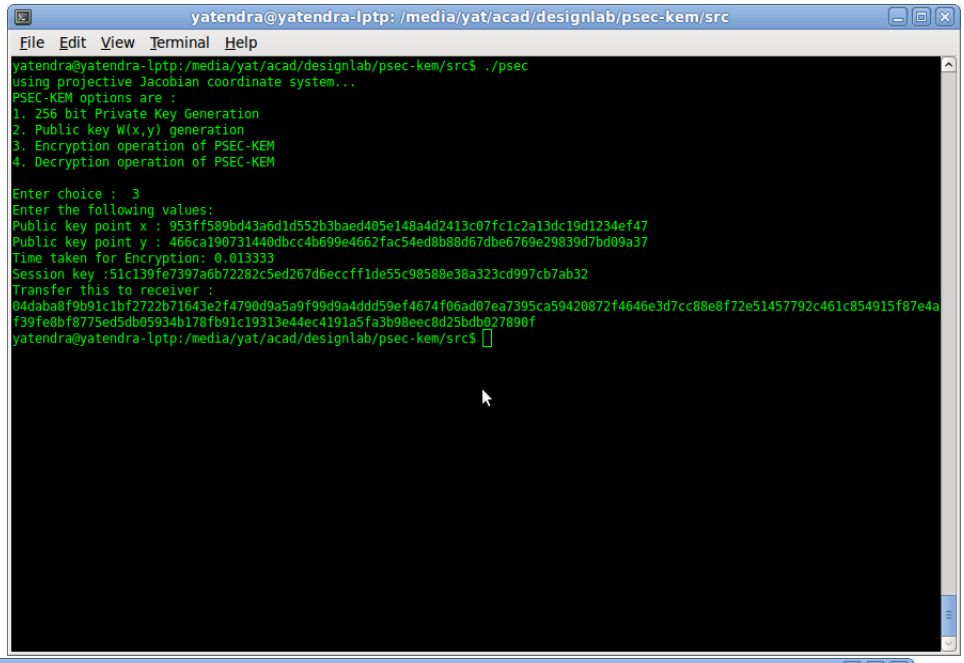
1. Store the values in the global variables of `psec-kem.h` namely `A, B, p` i.e. the curve and field parameters
2. Convert `A` and `B` to octet strings
3. Allocate space fot the various parameters like the session-key, public key point `W`, base point `P` etc.
4. Set the base point `P`.
5. Input an option from the user. Based on this four cases are possible.
   - For private key generation, print a random secret key on the terminal.
   - For public key generation, input the secret key `s` from user and print the point `sP`.
   - For encryption, input the public key and generate a session key and a message to be send to decryption end.
   - For decryption, input the secret key, public-key and the message from encryption end. Ouput session key if the decryption succeeds, else print `INVALID!` message

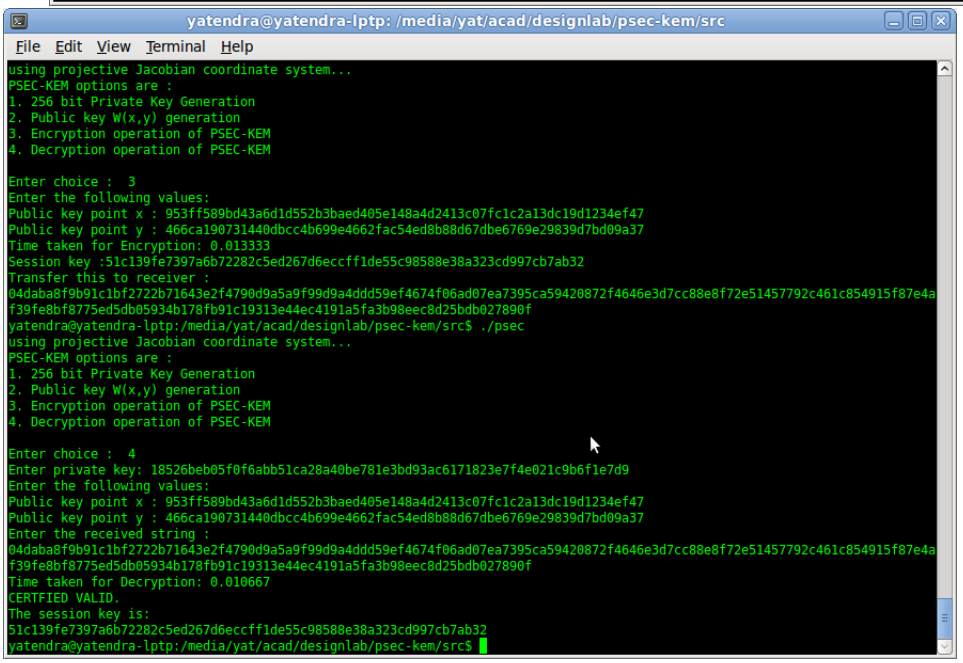
©NTT Corporation, IIT Kharagpur, 2012

5 Screenshots






yatendra@yatendra-lptp: /media/yat/acad/designlab/psec-kem/src
File Edit View Terminal Help
yatendra@yatendra-lptp:/media/yat/acad/designlab/psec-kem/src$ ./psec
using projective Jacobian coordinate system...
PSEC-KEM options are :
1. 256 bit Private Key Generation
2. Public key W(x,y) generation
3. Encryption operation of PSEC-KEM
4. Decryption operation of PSEC-KEM
Enter choice : 3
Enter the following values:
Public key point x : 953ff589bd43a6d1d552b3baed405e148a4d2413c07fc1c2a13dc19d1234ef47
Public key point y : 466ca190731440dbcc4b699e4662fac54ed8b88d67dbe6769e29839d7bd09a37
Time taken for Encryption: 0.013333
Session key :51c139fe7397a6b72282c5ed267d6eccff1de55c98588e38a323cd997cb7ab32
Transfer this to receiver :
04daba8f9b91c1bf2722b71643e2f4790d9a5a9f99d9a4ddd59ef4674f06ad07ea7395ca59420872f4646e3d7cc88e8f72e51457792c461c854915f87e4a
f39fe8bf8775ed5db05934b178fb91c19313e44ec4191a5fa3b98eec8d25bdb027890f
yatendra@yatendra-lptp:/media/yat/acad/designlab/psec-kem/src$


yatendra@yatendra-lptp: /media/yat/acad/designlab/psec-kem/src
File Edit View Terminal Help
using projective Jacobian coordinate system...
PSEC-KEM options are :
1. 256 bit Private Key Generation
2. Public key W(x,y) generation
3. Encryption operation of PSEC-KEM
4. Decryption operation of PSEC-KEM
Enter choice : 3
Enter the following values:
Public key point x : 953ff589bd43a6d1d552b3baed405e148a4d2413c07fc1c2a13dc19d1234ef47
Public key point y : 466ca190731440dbcc4b699e4662fac54ed8b88d67dbe6769e29839d7bd09a37
Time taken for Encryption: 0.013333
Session key :51c139fe7397a6b72282c5ed267d6eccff1de55c98588e38a323cd997cb7ab32
Transfer this to receiver :
04daba8f9b91c1bf2722b71643e2f4790d9a5a9f99d9a4ddd59ef4674f06ad07ea7395ca59420872f4646e3d7cc88e8f72e51457792c461c854915f87e4a
f39fe8bf8775ed5db05934b178fb91c19313e44ec4191a5fa3b98eec8d25bdb027890f
yatendra@yatendra-lptp:/media/yat/acad/designlab/psec-kem/src$ ./psec
using projective Jacobian coordinate system...
PSEC-KEM options are :
1. 256 bit Private Key Generation
2. Public key W(x,y) generation
3. Encryption operation of PSEC-KEM
4. Decryption operation of PSEC-KEM
Enter choice : 4
Enter private key: 18526beb05f0f6abb51ca28a40be781e3bd93ac6171823e7f4e021c9b6f1e7d9
Enter the following values:
Public key point x : 953ff589bd43a6d1d552b3baed405e148a4d2413c07fc1c2a13dc19d1234ef47
Public key point y : 466ca190731440dbcc4b699e4662fac54ed8b88d67dbe6769e29839d7bd09a37
Enter the received string :
04daba8f9b91c1bf2722b71643e2f4790d9a5a9f99d9a4ddd59ef4674f06ad07ea7395ca59420872f4646e3d7cc88e8f72e51457792c461c854915f87e4a
f39fe8bf8775ed5db05934b178fb91c19313e44ec4191a5fa3b98eec8d25bdb027890f
Time taken for Decryption: 0.010667
CERTFIED VALID.
The session key is:
51c139fe7397a6b72282c5ed267d6eccff1de55c98588e38a323cd997cb7ab32
yatendra@yatendra-lptp:/media/yat/acad/designlab/psec-kem/src$