An Example of Using the ECC-LIB

Elisavet Konstantinou* Yiannis Stamatiou † Christos Zaroliagis †

August 1, 2002

We illustrate the simplicity and ease of use of ecc-lib by presenting a small program that generates a digital signature using the Elliptic Curve Digital Signature Algorithm (ECDSA).

A digital signature is a pair of integers (r, s) whose values depend both on the message that is to be signed and the private key of the signer. The receiver of the signature must be able to verify the message. A very popular scheme that can be used for creating an elliptic curve signature protocol is the Elliptic Curve Digital Signature Algorithm. In order to create the signature, the user must firstly generate the prime order p of the finite field, the elliptic curve and its order m. Secondly, the user must create a base point which is used to generate a pair of keys (private and public). In particular, only the private key will be used in the creation of the signature. The function which will actually produce the signature takes as input the computed order p, the elliptic curve, the large prime factor n of m, the base point, the private key, and of course the message (or the file) to be signed. The details of the generation of a signature over a file are explained in the following simple program.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "gmp.h"

#include "cm.h"
#include "ec_operations.h"
#include "protocols.h"

gmp_randstate_t stat;

int main()
{
 /* Initialize all the necessary variables */
 /* Initialize the seed for the random number generator */
 /* Create the elliptic curve parameters */
 /* Sign the file */
 /* Clear all the variables */

 return 0;
}
```

*Computer Technology Institute and Department of Computer Engineering & Informatics, University of Patras, 26500 Patras, Greece. E-mails: {konstane,stamatiu,zaro}@ceid.upatras.gr.

The files `cm.h`, `ec_operations.h` and `protocols.h` are included in the program, since they contain the functions of ecc-lib that will be used in the program. Also, the file `gmp.h` must be included because the GNUMP library is used for the computations. Finally, the variable `stat` is defined globally as it is used by all functions that generate random numbers.

The ECDSA program consists of five parts of the program that are described in the following.

The first part (initialization step) is implemented as follows:

```
/* Initialization */
long D, sd = 0;

mpz_t p, m, n, h, private_key;
mpz_t curv[2], base_point[2], public_key[2];
mpz_t seed;

gmp_randinit(stat, GMP_RAND_ALG_LC, 120);

mpz_init(p);
mpz_init(m);
mpz_init(n);
mpz_init(h);
mpz_init(private_key);
mpz_init(curv[0]); mpz_init(curv[1]);
mpz_init(base_point[0]); mpz_init(base_point[1]);
mpz_init(public_key[0]); mpz_init(public_key[1]);
mpz_init(seed);
```

The variables `p,m` and `private_key` store the field's order, the EC's order m, and the private key of the signer, respectively, while the variables `curv`, `base_point` and `public_key` store the elliptic curve's parameters a and b (in `curv[0]` and `curv[1]`, resp.), the coordinates (x, y) of the base point of the EC (in `base_point[0]` and `base_point[1]`, resp.), and the public key of the signer. All the variables are initialized with the function `mpz_init()` of the GNUMP library, while the random state variable `stat` is initiallized with the function `gmp_randinit()`.

The second part concerns the assignment of an initial value to the random state variable `stat` with the help of a seed:

```
/* Initialize the random state variable stat */
srand( (unsigned) getpid());
sd=rand();
mpz_set_ui(seed, sd);

gmp_randseed(stat, seed);
```

The function `gmp_randseed()` sets the initial seed value, generated by the ANSI C function `rand()`.

The third part concerns the creation of the elliptic curve parameters.

```
/* set the value of the discriminant*/
D = 40;

/* create the order p, the elliptic curve and its order m using D */
CMmethod(D, &p, &m, curv);

/* create a base point of order n, where m = nh and returns n and h*/
domain_parameters(curv, base_point, &p, &m, &n, &h);

/* create a private and a public key */
create_priv_and_public(curv, &p, base_point, &private_key, public_key);
```

The function `CMmethod()` (defined in `cm.h`) takes as input the discriminant D (we gave it the value 40) and produces the field's order p, the parameters a and b of the EC, and the order m of the EC. The function `domain_parameters()` (defined in `ec_operations.h`) constructs the base point of the EC which has order n, where n is the large prime factor of m, and computes the two factors n and h. Finally, the function `create_priv_and_public()` (defined in `ec_operations.h`) produces the public and private keys of the signer based on the constructed EC, the field's order p, and the computed base point.

The fourth part concerns the generation of the signature on the file `README`.

```
/* generate the signature on the file README using ECDSA
   the signature is stored in the file signature */
  file_create_signature("README", curv, base_point, &p, &n, &private_key, "signature");
```

The function `file_create_signature()` (defined in `protocols.h`) generates the requested pair of integers (r, s) (and store them in a file) using the ECDS Algorithm, taking as input the file to be signed, the EC, p, the base point, the prime factor of m, and the private key of the signer.

The fifth part of the program clears the used variables which is requested by the GNUMP library.

```
gmp_randclear(stat);
mpz_clear(p);
mpz_clear(m);
mpz_clear(n);
mpz_clear(h);
mpz_clear(private_key);
mpz_clear(curv[0]); mpz_clear(curv[1]);
mpz_clear(base_point[0]); mpz_clear(base_point[1]);
mpz_clear(public_key[0]); mpz_clear(public_key[1]);
mpz_clear(seed);
```