Univ. Grenoble Alpes/Grenoble INP **2016–2017**
M2 Cybersecurity

Practical Work 1 (20 October 2016)

☞ *Beware : the following questions require a significant amount of work... !*

Practical Work on RSA

1 Follow the instructions given in the classroom in order to compile yourself `GMP` and the reference version (for the M2 Cybersecurity) of `PARI/GP`. In the remaining, activate the timer in GP.

2 Create a function `GeneratePseudoPrime(b)` which generates a (pseudo) prime number of exactly b bits.

3 Create a function `GenerateStrongPseudoPrime(b)` which generates a (pseudo) prime number p of exactly b bits such that $(p-1)/2$ is also (pseudo) prime. Compare the timing of both functions when the size of b increase. Is it easy to generate (pseudo)prime of 2048 bits ?

4 Create a function `GenerateRSAKey(b,e=`$2^{16}+1$`,flag=0)` which return a 7-tuple $[N, p, q, e, d, d_p, d_q]$, with N an RSA modulus of b bits, p and q (pseudo)primes such as $p < q < 2p$, with d the private exponent close to the size of N and d_p (resp. d_q) equals $d \mod p-1$ (resp. $d \mod q-1$). You can fix e in the input but the default value will be $2^{16}+1$. If `flag=1` then create RSA parameters using strong (pseudo)primes and else generic (pseudo)primes. Generate RSA parameters for modulus size of 512, 1024, 2048, 3072, 4096 and 8192 bits. Compare the timings w.r.t. the size.

5 Create functions `RSAencrypt(N,e,M)`, `RSAdecrypt(N,d,C)` and `RSAdecryptCRT(p,q,dp,dq,C)` for encryption, classical decryption and decryption using the CRT. We assume that M and C are integers with $0 \leqslant M, C < N$. The output of the functions should be integers. Compare the decryption time with CRT w.r.t. without CRT for RSA modulus of size 1024,2048,4096 and 8192 bits.

Practical Work on ECC

The main functions for manipulating elliptic curves with PARI/GP are : `ellinit, elladd, ellsub, ellmul, ellorder, ellordinate, ellisoncurve`.
Let investigate these functions :
`E=ellinit(x)` ; where x is the vector of Tate coefficients for the curve E, namely $x = [a_1, a_2, a_3, a_4, a_6]$ defining the curve

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6\,.$$

The function `ellinit` will initialise the curve E and return a large vector. The field where the coefficients a_i are defined will set the base field for the curve. If $a_i \in K$ then E will be a curve over K. Furthermore the point `[0]` represents the infinity. We can access the data of E as member functions. For instance
`E.j` will return the j-invariant, `E.disc` its discriminant, `E.b4` the Tate b_4 coefficient, etc.
It is also possible to define E from a short Weierstrass form by giving as input the pair $[a, b]$.
If P and Q are two points on the curve E, then `R=elladd(E,P,Q)` ; will compute $R = P + Q$ on the curve E. In the same way `ellsub(E,P,Q)` ; will compute $P - Q$.
The function `ellmul(E,P,n)` ; with P on the curve E and n a positive integer will compute nP.

For instance computing $2P$ (i.e., doubling) can be done more efficiently than computing $P+P$.
The function `ellorder(E,P)` will compute the order of P. If P is non-torsion (*never the case when the curve is defined over a finite field*), the function return 0.
The function `ellordinate(E,x)` will return the y-coordinates of E corresponding to the ordinate x. If x is not the x-coordinate of an affine point of E, the function will return the infinity (i.e., `[0]`) or an empty set.
The function `ellisoncurve(E,P)` returns 1 (i.e., true) if P is on the curve E and 0 if not.

Recall that the Frobenius trace of an EC over $\mathbb{F}_p$ is defined as $a_p = p+1-\#E(\mathbb{F}_p)$. The function `ellap(E,p)` computes the Frobenius trace (at p) of E (defined over $\mathbb{F}_p$) Notice that p *must be a prime*. Not all the PARI/GP functions for elliptic curves work with non-prime finite fields (and in particular `ellap`.
Notice that the function uses Shanks/Mestre algorithm for small p (less than 30 digits)
and Schoof/Elkies/Atkin for larger primes (up to 200 digits).

1 Familiarise yourself with the PARI/GP functions for ECC.

2 Using the "Microsoft EC", and let p the defined prime. Check that p is prime (not pseudo-prime !). Check the order of $E(\mathbb{F}_p)$. Check that P is on the curve and that its order correspond to $\#E(\mathbb{F}_p)$.

3 For all the NIST curves (see the file `ECC_NIST.gp`), check that the "prime numbers" are indeed prime. Compute $\#E(\mathbb{F}_p)$ for each curve and check that P has order $\#E(\mathbb{F}_p)$.

4 Using the "Apple EC" (aka Montgomery representation), construct a program which given a prime number p generate an EC of "Apple type" with $\#E(\mathbb{F}_p)$ of the form $4q$ with q prime. Generate curves of 256 bits and 512 bits.

5 Create a program that given a prime p generate an EC over $\mathbb{F}_p$ with Weierstrass equation of the type $y^2 = x^3 + x + b$. Generate 1000 curves and look at the range of $\#E(\mathbb{F}_p)$ w.r.t. the Hasse bounds seen in the lectures.

6 Create a program that given a prime p generate an EC over $\mathbb{F}_p$ with Weierstrass equation of the type $y^2 = x^3 + x + b$ and such that $\#E(\mathbb{F}_p)$ is either a prime number or of the form $4q$ with q a prime number. In the last case, the program should also return the 2-torsion points of the curve. Compare the timings.